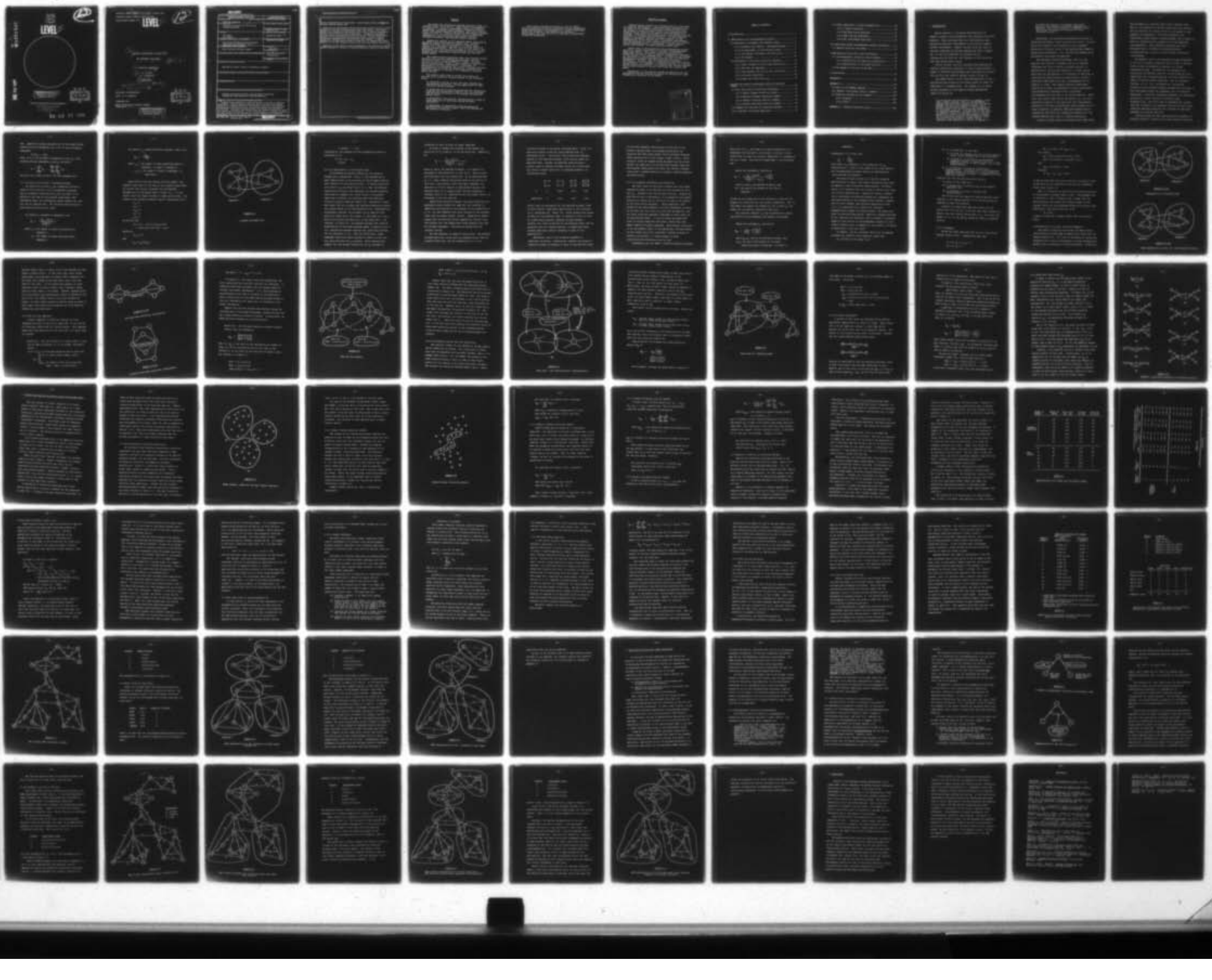
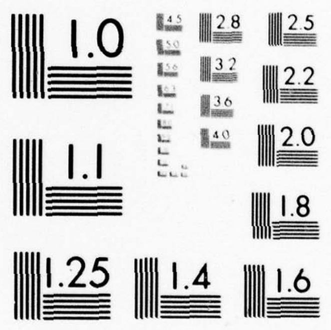
AD-A071 647 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
ANALYSIS TECHNIQUES FOR USE WITH THE EXTENDED SDM MODEL.(U)
MAY 79 S L HUFF, S E MADNICK N00039-78-G-0160
UNCLASSIFIED CISR-P010-7905-09 NL

1 OF 2
AD
A071647





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A071647

Center for Information Systems Research

Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

15

Contract Number N00039-78-G-0160 (Order 002)

Internal Report Number P010-7905-09

Deliverable Number 002 **LEVEL** II

12

6

ANALYSIS TECHNIQUES FOR USE WITH THE EXTENDED SDM MODEL.

12 124p.

9 Technical Report, 9

10 S. L. Huff
S. E. Madnick

11 May 1979

14 CISR-P010-7905-09, CISR-TR-9

Principal Investigator:

Prof. S. E. Madnick

Prepared for:

Naval Electronic Systems Command
Washington, D. C.

JOB

79 07 19 001

409 590

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #9	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "Analysis Techniques for Use With The Extended SDM Model"		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER P010-7905-09
7. AUTHOR(s) S.L. Huff S.E. Madnick		8. CONTRACT OR GRANT NUMBER(s) N00039-78-G-0160
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center For Information Systems Research Sloan School of Management, M.I.T. Cambridge, Mass. 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Electronic Systems Command		12. REPORT DATE May 1979
		13. NUMBER OF PAGES 115
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software architectural design; problem design structuring; mathematical graph modelling; graph decomposition.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional requirements into sub-problems to form a design structure that will exhibit

next Page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the key characteristics of good design: strong coupling within sub-problems, and week coupling between them.

Recent work in the Systematic Design Methodology project has led to certain extensions to the basic representational model used therein. This report presents new analytical mechanisms that may be used to execute decomposition analyses in the context of the extended model. Included are new methods for calculating inter-requirement similarities, for measuring decomposition goodness, and for generating clustering hierarchies. As well, some new hierarchical clustering tools are presented, with examples. A complete analysis of a 22-node design problem is presented, and compared with results obtained for the same problem in earlier work using the original SDM model.

Appendices of this report include documentation of the interactive analysis package developed for carrying out graph decomposition analysis on SDM models.

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management. It consists of a group of management information systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, implementing, and maintaining application software, information systems, and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-78-G-0160, CISR has proposed to conduct basic research on a systematic approach to the early phases of complex systems design. The main goal of this work is the development of a well-defined methodology to fill the gap between system requirements specification and detailed system design.

The research being performed under this contract builds directly upon results stemming from previous research carried out under contract N00039-77-C-0255. The main results of that work include a basic scheme for modelling a set of design problem requirements, techniques for decomposing the requirements set to form a design structure, and guidelines for using the methodology developed from experience gained in testing it on a specific, realistic design problem.

The present study aims to extend and enhance the previous work, primarily through efforts in the following areas:

1) additional testing of both the basic methodology, and proposed extensions, through application to other realistic design problems;

2) investigation of alternative methods for effectively coupling this methodology together with the preceding and following activities in the systems analysis and design cycle;

3) extensions of the earlier representational scheme to allow modelling of additional design-relevant information;

4) development of appropriate graph decomposition techniques and software support tools for testing out the proposed extensions.

This report pertains to points (3) and (4) above. Various extensions to earlier Systematic Design Methodology graph representation and decomposition analysis techniques, as well as a number of new methods, are presented. Some examples are carried out to illustrate the usefulness of the new techniques.

EXECUTIVE SUMMARY

Complex design problems are characterized by a multitude of competing requirements. System designers frequently find the scope of the problem beyond their conceptual abilities, and attempt to cope with this difficulty by decomposing the original design problem into smaller, more manageable sub-problems. Functional requirements form a key interface between the users of a system and its designers. In this research effort, a systematic approach has been proposed for the decomposition of the overall set of functional requirements into sub-problems to form a design structure that will exhibit the key characteristics of good design: strong coupling within sub-problems, and week coupling between them.

Recent work in the Systematic Design Methodology project has led to certain extensions to the basic representational model used therein. This report presents new analytical mechanisms that may be used to execute decomposition analyses in the context of the extended model. Included are new methods for calculating inter-requirement similarities, for measuring decomposition goodness, and for generating clustering hierarchies. As well, some new hierarchical clustering tools are presented, with examples. A complete analysis of a 22-node design problem is presented, and compared with results obtained for the same problem in earlier work using the original SDM model.

Appendices of this report include documentation of the interactive analysis package developed for carrying out graph decomposition analysis on SDM models.

TABLE OF CONTENTS

1 Introduction.

Recent advances in the design and manufacture of computer hardware, and accompanying cost decreases, have been dramatic. These successes have not, unfortunately, been matched by equivalent improvements in the area of computer software development. Despite the goal first put forth at the Nato Conference On Software Engineering (Naur & Randell 68), to turn the production of software into an engineering-like activity, real advances in this direction have come surprisingly slowly.

One of the most important problems being addressed by software engineering researchers is that of system design. A number of different methodologies and techniques for guiding this activity has recently emerged (Peters & Tripp 77).

One common feature of all these methodologies is that a preliminary partitioning of the target system is either not addressed or is assumed given. For example, Dr. G. Myers, primary developer of the Composite Design Methodology (Myers 78) points out

> "If the product being developed is a system, rather than a single program, there is another design process that must occur between the external design process and the use of composite design. This process, called system design, is the decomposition of the system into a set of individual subsystems or individual programs. Although some of the ideas of composite design are appropriate here, and some people have claimed to have used composite design for this process, composite design does not appear to be directly applicable to system design. Therefore, when designing a system, as opposed to

an individual program, the designer must first partition the system into distinct subsystems or programs. Then the methodology of composite design can be used to produce the structure of these individual pieces."

Nonetheless, the preliminary problem partitioning, or "architectural design," task is not at all trivial. In fact, one of the reasons it has received so little research attention in the past is that it has usually been viewed as analytically intractable - i.e., too deep and complex to be successfully structured and modelled.

The Systematic Design Methodology (SDM) is a new approach that provides a framework and set of analysis techniques and tools to assist a software designer in determining an architectural design for a target system. Underlying the SDM approach is a technique for modelling an architectural design problem by representing a system's functional requirements and their interdependencies as a network, or graph. The simplest possible graph model, consisting of nodes (corresponding to system requirements) interconnected by unweighted links (requirement interdependencies) was used in the initial SDM work. While this "bare bones" model proved satisfactory for the early exploratory studies, it was also clear that improvements, primarily in the form of extensions, could be made so as to allow a designer to represent additional design-relevant information, and bring this information usefully into the analysis methods that lead to a system architecture.

Certain of these potential extensions were identified and discussed in a previous report (Huff & Madnick 1978). There it was argued that the most significant such extension was the inclusion of a weight factor to correspond to each assessed interdependency. With this extension, the requirements graph becames a weighted graph, with a weight on each arc representing the strength of the corresponding interdependency. Other possible extensions were also discussed there, including relationships between interdependencies, as well as certain kinds of directed relationships.

An important part of the Systematic Design Methodology is the set of analysis techniques that are used to perform various kinds of decomposition analysis on a given requirements graph. Included here are procedures for calculating the goodness index for a given graph partition (based upon the "strength/coupling" criterion argued for by a number of software design theorists; see, for instance, (Stevens, et. al. 75)); procedures for calculating similarity and/or distance measures between pairs of nodes in the requirements graph, for subsequent use in clustering algorithms; clustering techniques themselves, for performing hierarchical cluster analysis to generate graph decompositions; and other types of decomposition analysis routines, including a new top-down hierarchical paritioning algorithm developed especially for treating the SDM graph decomposition problem (Huff 79).

Having extended the SDM representational framework, it becomes necessary to modify the various analysis techniques so as to incorporate the information included in the expanded representation. The major purpose of this paper is to present, justify and discuss certain new analytical mechanisms that were developed to treat the requirements decomposition problem in the context of the extended SDM model. In addition, this report will present other new analytical techniques and discuss their pros and cons. Some potentially valuable decomposition approaches, which have not yet been fully exploited in the SDM context, will be outlined. The results of a comparative analysis among currently viable decomposition methods will be presented. Two different approaches to incorporating interdependency similarity information into the graph decompositon will also be briefly discussed. A medium-scale example analysis illustrating the effectiveness of the extended analysis techniques is presented, and contrasted to the somewhat different results obtained for the same example graph in earlier work.

Finally, the appendices of this report include documentation and an example execution trace of the computer package that has been developed to implement the new analysis methods.

2 SDM Analysis and Interdependency Weights.

As stated above, the single most useful and important extension made to the original requirements graph model involves incorporation of weight factors corresponding to the requirement interdependencies. In the work to date, weights are chosen from the range [0, 1.0], with lower values corresponding to "weaker" interdependencies. A useful practical device in this regard is to select the weights from three possible candidates: W (weak), A (average), or S (strong). For computational purposes, these are mapped into numerical values, for example,

S - 0.8

A - 0.5

W - 0.2 .

In this section we examine extensions that have been made to various analysis mechanisms to incorporate such link weights.

2.1 Extension to Decomposition Goodness Index.

The concept of decomposition goodness has been captured by quantifying a commonly accepted notion of software design quality: Alexander (Alexander 64), Stevens (Stevens 75), Myers (Myers 78) and others have convincingly argued that a good software design is one that consists of modules that possess high strength, or internal binding, and which simultaneously are weakly interconnected. In the SDM, this "strength/coupling" criterion is quantified in the following way. Suppose the graph representation of the target design problem has been decomposed into a set of non-overlapping subgraphs:

$\{G_1, G_2, \ldots, G_k\}$.

Then, if S_i = the strength of subgraph G_i, and C_{ij} = the coupling between subgraphs G_i and G_j, we define

$$M = \sum_{i=1}^{k} S_i - \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} C_{ij}$$

and use M as a figure of merit for the decomposition.

2.1.1 Strength and Coupling - Unweighted Graphs.

The quantities S_i and C_{ij} are themselves defined in terms of the structure of the corresponding subgraphs. Various arguments regarding how S_i and C_{ij} ought to be defined in the case of the original graph model (with unweighted links) are discussed by Andreu (Andreu 78), and will not be repeated here. The following definitions for these quantities were given there:

(1) define S_i (strength of subgraph i) as:

$$S_i = \frac{L_i - (n_i - 1)}{\frac{n_i(n_i - 1)}{2}}$$

where L_i = the number of links contained within subgraph i,

n_i = the number of nodes contained within subgraph i.

(2) define C_{ij} (coupling between subgraph i and j) as:

$$C_{ij} = \frac{L_{ij}}{n_i n_j}$$

where L_{ij} = the number of links connecting nodes in subgraph i to nodes in subgraph j,

n_i, n_j = the number of nodes in subgraphs i,j spectively.

These definitions have strong intuitive appeal, and seemed to work well in the case of the original graph model. To clarify them further, consider the two-subgraph decomposition of the graph shown in Figure 2.1. In that figure, the total graph includes 11 nodes and 16 links. The values of the various parameters used in calculating M are:

$L_1 = 6$

$L_2 = 7$

$L_{12} = 3$

$n_1 = 5$

$n_2 = 6$

So we find that

$$S_1 = (L_1 - (n_1-1))/(n_1(n_1-1)/2)$$
$$= (6-(5-1))/(5(5-1)/2) = 0.20 .$$

Similarly,

$$S_2 = 0.13 .$$

And,

$$C_{12} = L_{12}/(n_1 n_2)$$

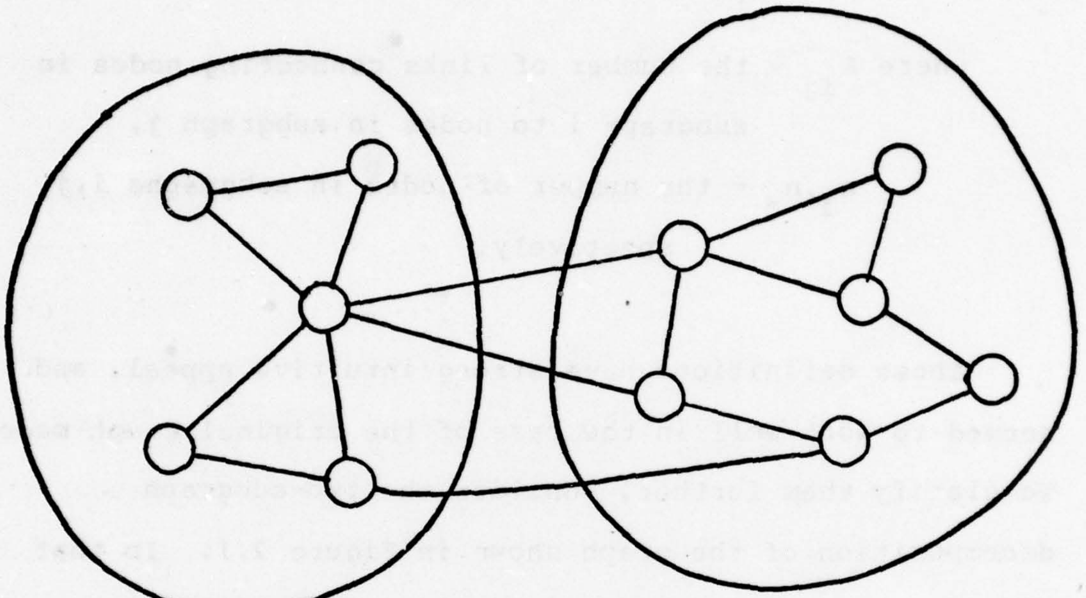


Figure 2.1

A simple decomposition.

$= 3/(5(6)) = 0.10$.

Consequently, the goodness of this decomposition would be calculated to be

$$M = S_1 + S_2 - C_{12}$$
$$= 0.230$$

2.1.2 An Improvement to the Strength Index.

In extending the measure definition, two kinds of changes were incorporated. First, it was decided that a small modification to the structure of the strength function would improve its value qualitatively. It may be noted that the value of C_{ij} can range from 0 to 1. Now, the software engineering literature that addresses the strength/coupling issue does not suggest that either factor is of primary importance in design, but indicates that both should be treated as equally important (Myers 78). Therefore it makes most sense that our strength and coupling formulations should carry equal weight in the determination of M. Unfortunately, as presently defined, the strength term S_i does not fall within the range [0,1], but rather, in the range [0, 1-2/n], while the coupling term does fall in the range [0,1]. For instance, in Figure 2.1, the maximum strength that could be exhibited by the five-node subgraph number 1 would be 1-2/5 = 0.6., whereas the maximum coupling that could occur between the two subgraphs is 1.0. From this definition, larger subgraphs would have higher maximum S_i values, as the term $2/n_i$ would decrease as n_i increases. These observations suggest that the present formulation for S_i includes an (unwanted) an bias in favor of larger subgraphs.

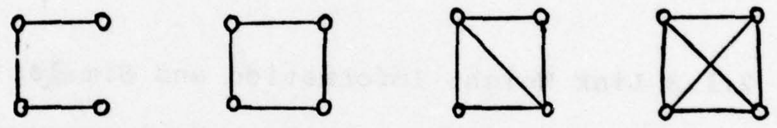
In order to remedy this problem, we may adjust the definition of S_i slightly, in the following way. Redefine S_i as:

$$S_i = \frac{L_i - (n_i - 1)}{\frac{n_i(n_i-1)}{2} - (n_i - 1)}$$

where L_i and n_i are defined as before. The range of S_i according to this definition is then [0,1]. There is only one difficulty with this definition: S_i is undefined when $n_i = 2$. Thus a special calculation must be carried out in this case. Fortunately, since very small subgraphs are generally of little interest in SDM analysis anyway, the approach taken is to simply assign two-node subgraphs a strength value of 1.0 (modified by the link weight factor, as discussed in the next section).

Andreu did originally consider the above definition for subgraph strength (Andreu 78, page 102) but rejected it, apparently only because of the presence of the singularity at $n = 2$. He also commented that both versions of the strength index tend to produce similar results. Essentially, Andreu argued that the modified version appeared no "better" than the original one, and complicated things slightly (the case of two-node subgraphs). Therefore he worked with the original version.

We take somewhat the opposite stance here: the modified strength definition given above <u>is</u> arguably better than the original definition, and the singularity at $n = 2$ is trivially avoided in the manner discussed above. First, the modified strength avoids the large-subgraph bias, as discussed earlier. Intuitively, fully-connected subgraphs should all have maximum strength, which they do under the modified index. Second, the modified index provides a greater range of possible strength values for a given node set, hence a higher sensitivity to subgraph geometry, as illustrated below:



S	0	0.167	0.33	0.50
modified S	0	0.33	0.67	1.00

Third, there are precedents for the modified strength index in other similar graph model applications in the literature (e.g., Estabrooke 66; Hubert 74). These authors have argued in favor of the modified index, in contexts similar to ours, as a good general-purpose subgraph strength measure. Finally, the "nice" properties of the modified index, and the parallels with the coupling measure, as summarized below, provide additional indirect evidence in favor of the modification.

Admittedly, none of the foregoing arguments is completely conclusive. Nonetheless, together they present a cumulative weight of evidence in favor of the modified index.

In the final analysis, choices such as this one in the present research effort are perhaps guided better by intuition and "what makes sense" than by provable theorems. (This characteristic is not unique to SDM, either.) While trying to locate an example graph decomposition that would "prove" the superiority of the modified strength index over the original index proved fruitless, the indirect evidence cited above is deemed substantial enough to warrant adopting the modification.

2.1.3 Link Weight Information and Similarity.

The other issue at this point concerns how link weight information ought to be factored into the calculation of S_i and C_{ij}. Consider first the strength function. Perhaps the most obvious way of extending S_i to incorporate link weights would be to replace the L_i term with the sum of the weights on the links within subgraph i. While appealing, this definition has some drawbacks, the most signigicant of which is the fact that the value of S_i may then be negative, even for fully connected subgraphs (subgraphs in which there are no disconnected nodes). It may of course be argued that there is nothing especially bad about a definition that admits negative strength subgraphs. Nevertheless, since most of our development effort in the SDM project has been guided by what seems intuitively reasonable (lacking criteria of obsolute correctness), it seems prudent to avoid counterintuitive definitions such as this.

Consequently we will adopt a slightly different extended definition of S_i. We accept the original definition as a reasonable starting point, and attempt to extend it while maintaining its positive featives, especially its residing in the [0,1] range. This may be accomplished in the following way:

Define the extended S_i function as

$$S_i' = \frac{L_i - (n_i - 1)}{\frac{n_i(n_i-1)}{2} - (n_i - 1)} * \left(\frac{W_i}{L_i}\right)$$

where L_i and n_i are defined as before, and

W_i = the sum of the weights on the links in subgraph i.

S_i' has all the properties of the original S_i, plus the new property of reflecting the link weight information. That is, the higher the summed link weights W_i (for a given L_i) the higher S_i', as would be required by intuition.

In a parallel fashion, C_{ij} may be modified to capture the effect of inter-subgraph link weights, as follows.

Define the extended C_{ij} function as

$$C_{ij}' = \frac{L_{ij}}{n_i n_j} * \left(\frac{W_{ij}}{L_{ij}}\right)$$

where L_{ij}, n_i, and n_j are defined as before, and

W_{ij} = the sum of the weights on the links connecting nodes in subgraph i to nodes in subgraph j.

Consequently it is clear that

$$C'_{ij} = \frac{W_{ij}}{n_i n_j} .$$

Once again, C'_{ij} possesses all the properties of C_{ij} (specifically, it falls in the range [0,1]), plus captures the interdependency strength effects as represented by inter-subgraph link weights.

The question again arises as to why the foregoing modifications to capture link weight information were used, as opposed to some alternative modifications. As before, there is no "theorem" that can be used to "prove" that these are the best modifications. Cumulative indirect evidence (e.g., maintenance of desirable properties, widespread use and understanding of the arithmetic mean concept), together with our best jusdment and favorable experience to date (i.e., no counterintuitive results for "obvious" decompositions) support the above choices. One additional point is also worth mentioning. Empirical evidence with SDM so far indicates that designers tend to produce a reasonably symmetric distribution of interdependency weights, so that using arithmetic mean, as opposed to, say, median, introduces no appreciable "long tail" bias.

In summary, the new (extended) definition of subgraph strength, S'_i, exhibits three important properties:

(1) it falls in the range [0,1];

(2) it is normalized, in two ways:

a) in terms of subgraph size (for a given number of links, larger subgraphs have lower strengths),

b) in terms of "tree-relative connectedness" (subgraph strength is measured relative to the minimum necessary to form a graph - i.e., "tree" connectedness);

(3) it is invariant in terms of "proportional connectedness": regardless of n, a tree-connected subgraph always has strength 0, a fully connected subgraph always has strength 1.0 (assuming all links have unity weight).

Similarly for the new coupling definition C'_{ij}:

(1) falls in the range [0,1],

(2) is normalized in terms of size of the coupled subgraphs, and

(3) is invarient in terms of "proportional connectedness."

The strong intuitive appeal of these properties lends credence to the appropriateness of the new definitions of subgraph strength and coupling.

Finally, one additional important feature of the definitions of S'_i and C'_{ij} is the fact that they include as a special case the original definitions (i.e., with each link weight set to 1.0, S'_i and C'_{ij} become S_i and C_{ij} as defined earlier).

2.1.4 An Example.

To see how these functions work out in a calculation, consider Figure 2.2(a). Computations show that

$$L_1 = 6,\ L_2 = 7,\ L_{12} = 3$$

$$n_1 = 5,\ n_2 = 6,$$

$W_1 = 3.2,\ W_2 = 3.8,\ W_{12} = 1.2$

As a result,

$S'_1 = (6-4)/(5(4)/2-4)*3.2/6 = 0.18$,

$S'_2 = (7-5)/(6(5)/2 - 5)*3.8/7 = 0.11$,

and

$C'_{12} = 1.2/(5(6)) = 0.04$.

Finally,

$M = S'_1 + S'_2 - C'_{12} = 0.25$.

It may be noted that this value turns out to be quite close to the value of 0.23 obtained in the earlier (unweighted links) case, Figure 2.1.

Now, to illustrate the sensitivity of the new functions to link weights, consider Figure 2.2(b). The decomposition is identical to that of Figure 2.2(a), except that the inter-subgraph link weights are slightly higher on the average, while the intra-subgraph weights are slightly lower than before. The new M turns out to be

$M = 0.32$,

a value, as expected, somewhat higher than the previous value.

2.2 Extensions to the Basic Similarity Measure.

The central analytical task within the SDM involves identification of good graph decompositions (those with the highest possible M). One class of techniques for generating decompositions involves transforming the graph decomposition problem into a hierarchical clustering problem. Clustering

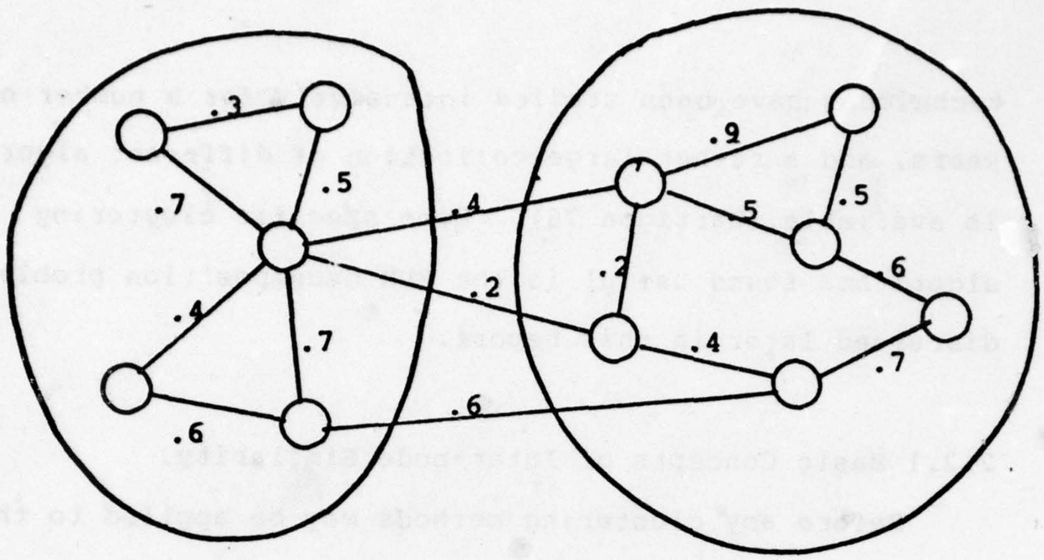


Figure 2.2 (a)

Decomposition of weighted graph.

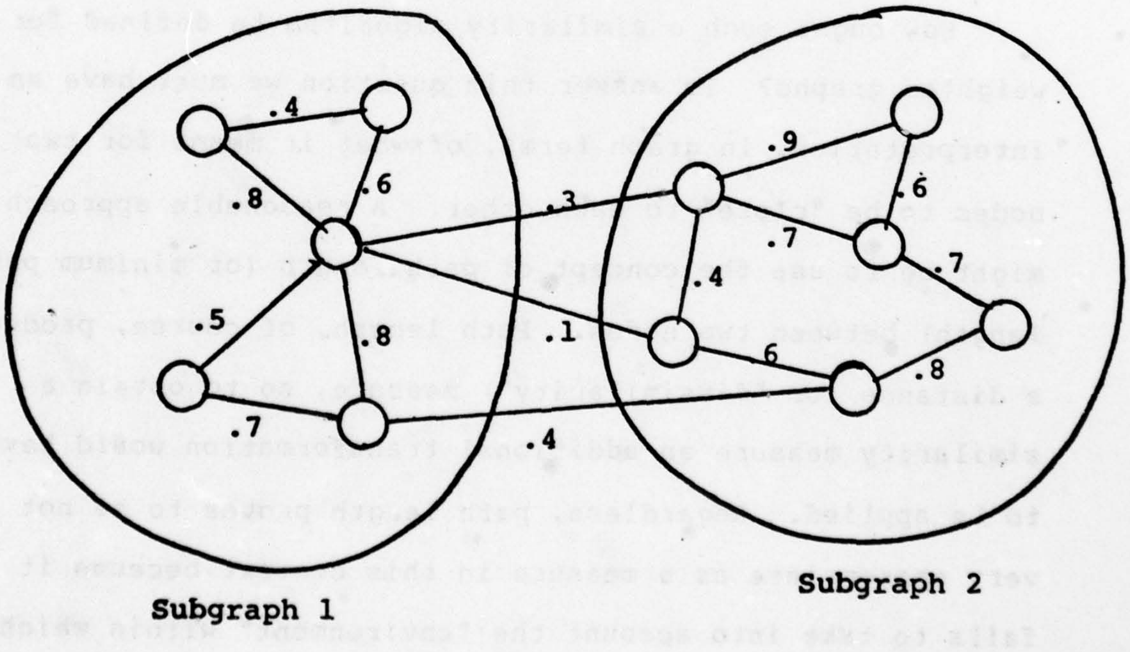


Figure 2.2 (b)

Same decomposition as for (a), with different weights.

techniques have been studied intensively for a number of years, and a rather large collection of different algorithms is available (Hartigan 76). Some specific clustering algorithms found useful is the SDM decomposition problem are discussed later in this report.

2.2.1 Basic Concepts of Inter-node Similarity.

Before any clustering methods may be applied to the SDM graph decomposition problem, a measure of "similarity," or closeness between each pair of nodes in the requirements graph must be defined. Basically, a similarity algorithm transforms a graph into a similarity matrix, which may be used to drive various clustering algorithms to produce a graph decomposition hierarchy. These relationships are illustrated in Figure 2.3.

How ought such a similarity algorithm be defined for weighted graphs? To answer this question we must have an interpretation, in graph terms, of what it means for two nodes to be "close" to each other. A reasonable approach might be to use the concept of path length (or minimum path length) between two nodes. Path length, of course, produces a distance (or "dissimilarity") measure, so to obtain a similarity measure an additional transformation would have to be applied. Regardless, path length proves to be not very appropriate as a measure in this context because it fails to take into account the "environment" within which a pair of nodes resides. For example, we would clearly want a good similarity algorithm to produce a lower similarity

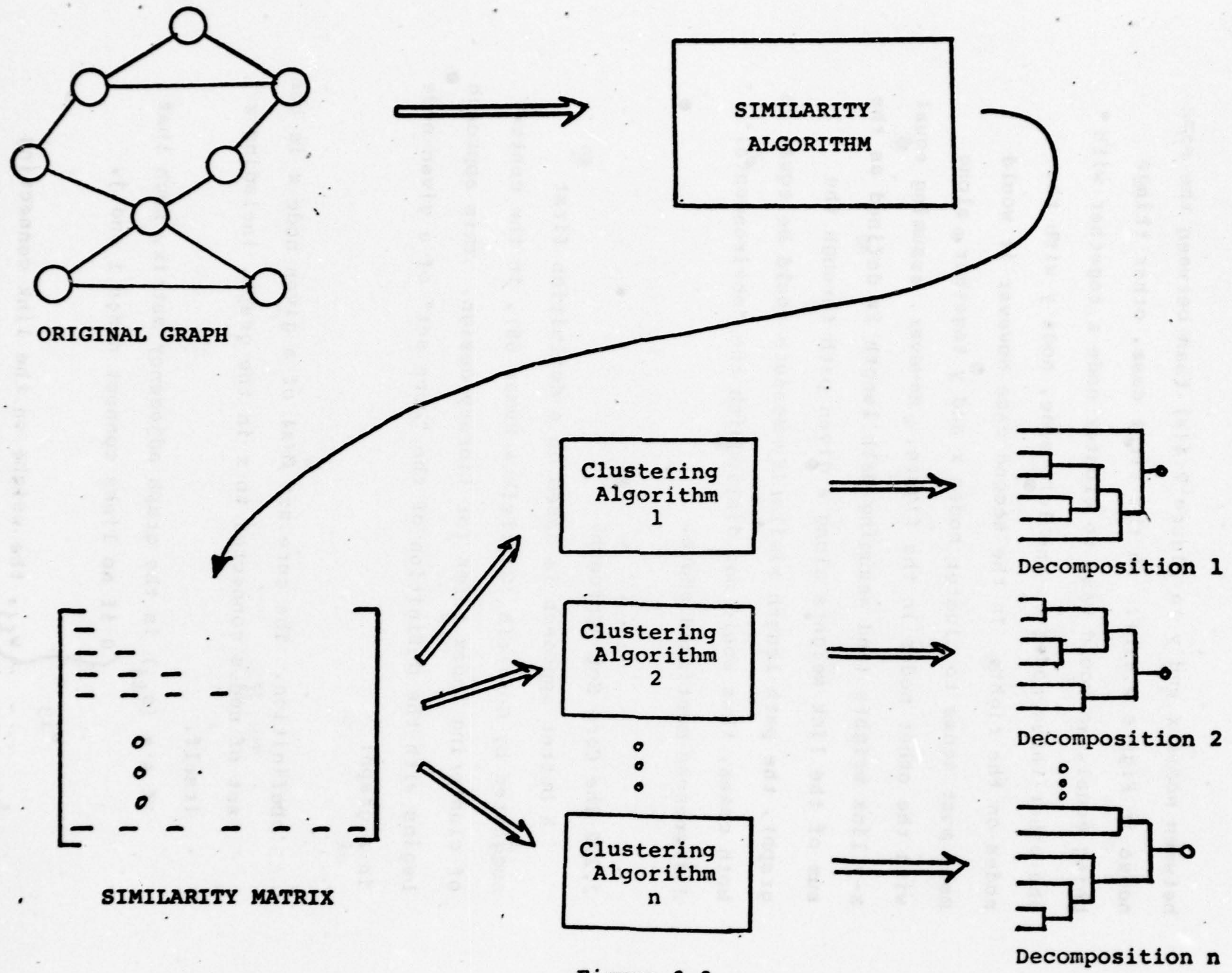


Figure 2.3

Graph decomposition via cluster analysis - the process.

between nodes x and y in Figure 2.4(a) than between the same nodes in Figure 2.4(b). In the first case, other things being equal, we would want to cluster node x together with the other three nodes on the left side, node y with the nodes on the right. In the second case however it would make most sense to cluster nodes x and y together, along with the other nodes in the figure. However, assuming equal x-y link weights (and assuming path length is defined as the sum of the link weights along a given path through the graph), the path length similarity measure would be equal in both cases, thus would not distinguish the "environmental" differences mentioned above.

2.2.2 The Core Set Approach.

A better approach is based on a definition first suggested by Gottleib (Gottleib & Kumar 68), in the context of clustering index terms for library design. This approach begins with the definition of the "core set" of a given node in a graph:

> Definition. The core set $\emptyset(x)$ of a given node x is the set of nodes connected to x in the graph, including x itself.
>
> If $A = (a_{ij})$ is the graph adjacency matrix, such that
>
> $$a_{ij} = \begin{cases} 0 \text{ if no links connect nodes } i \text{ and } j, \\ w_{ij}, \text{ the weight on the link connecting nodes } i \text{ and } j, \text{ if one exists,} \end{cases}$$

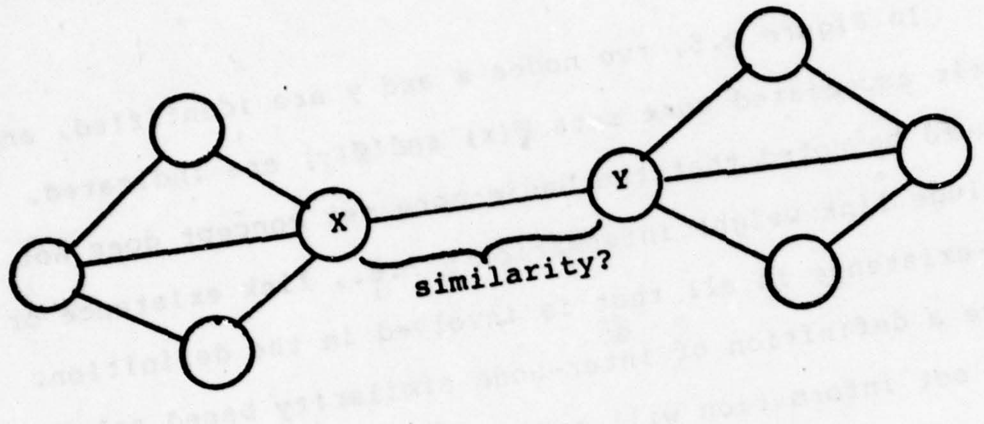


Figure 2.4 (a)

An inter-node similarity illustration.

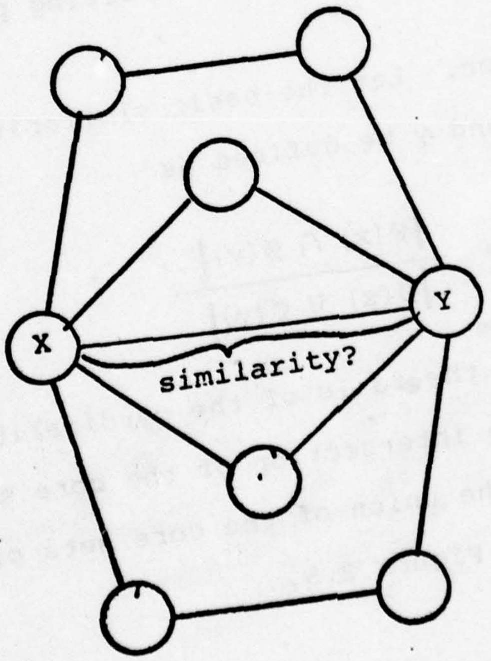


Figure 2.4 (b)

A second inter-node similarity illustration.

then $\emptyset(x) = \{ y: a_{xy} > 0 \} \cup \{x\}$.

In Figure 2.5, two nodes x and y are identified, and their associated core sets $\emptyset(x)$ and $\emptyset(y)$ are indicated. It should be noted that the basic core set concept does not include link weight information - i.e., link existence or non-existence is all that is involved in the definition. Hence a definition of inter-node similarity based solely on core set information will again fail to incorporate link weight information.

However the core set concept does provide a means for including part of the graph environment information that the path length definition failed to achieve. In particular, we adopt Gottleib's measure as a starting point:

Definition. Let the basic similarity measure between nodes x and y be defined as

$$P'_{xy} = \frac{|\emptyset(x) \cap \emptyset(y)|}{|\emptyset(x) \cup \emptyset(y)|}$$

That is, P'_{xy} is the ratio of the cardinality of (number of nodes in) of the intersection of the core sets to the cardinality of the union of the core sets of nodes x and y. For instance, in Figure 2.5,

$$\emptyset(x) = \{x,2,3,4,5,y\}$$

$$\emptyset(y) = \{y,x,5,6,7,8\}$$

$$|\emptyset(x) \cap \emptyset(y)| = |\{x,y,5\}| = 3$$

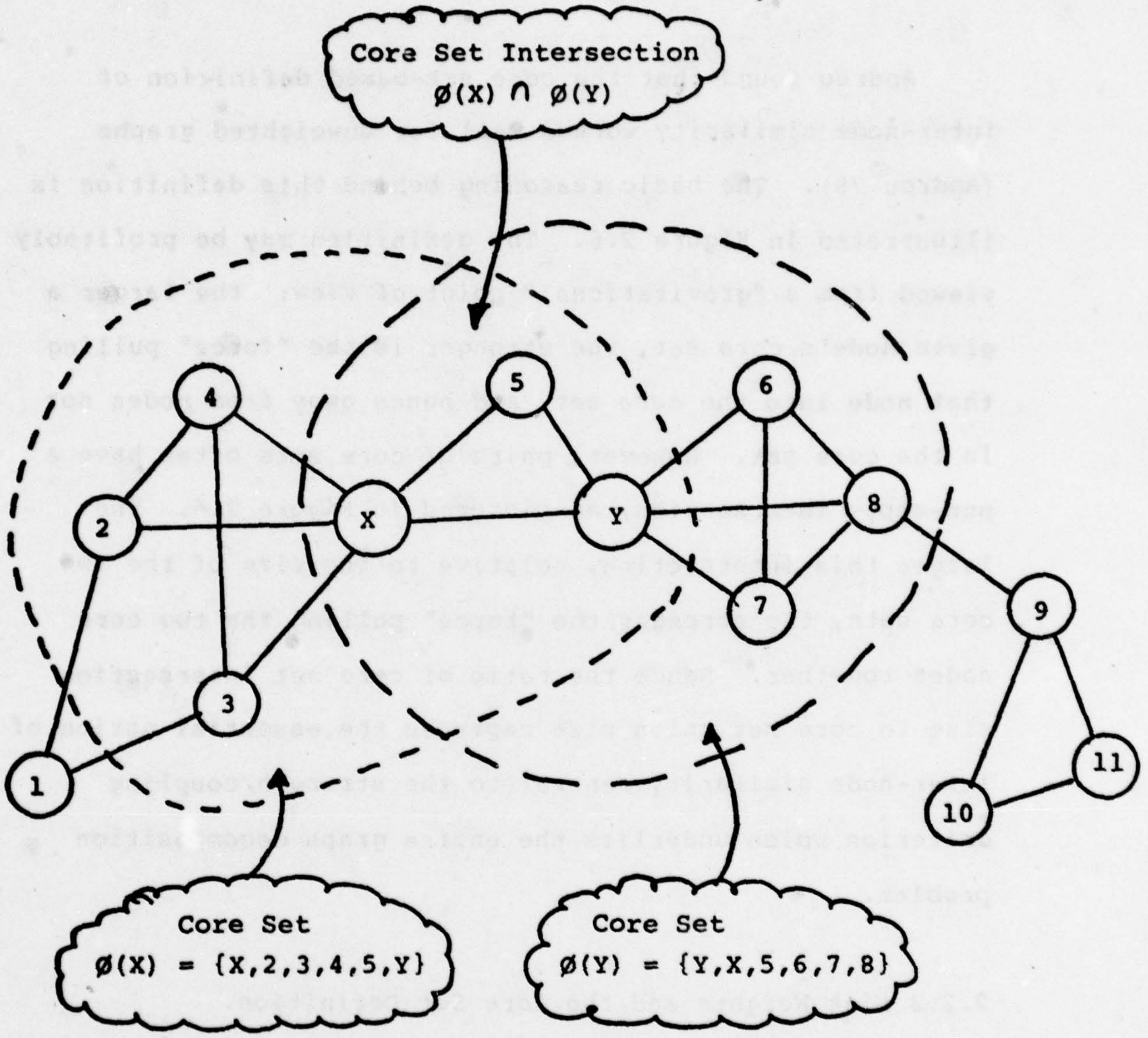


Figure 2.5

The core set concept.

$|\emptyset(x) \cup \emptyset(Y)| = |\{x,y,2,3,4,5,6,7,8\}| = 9$, so

$P'_{xy} = 3/9 = 0.33$.

Andreu found that the core set-based definition of inter-node similarity worked well for unweighted graphs (Andreu 78). The basic reasoning behind this definition is illustrated in Figure 2.6. The definition may be profitably viewed from a "gravitational" point of view: the larger a given node's core set, the stronger is the "force" pulling that node into the core set, and hence away from nodes not in the core set. However, pairs of core sets often have a non-empty intersection, as pictured in Figure 2.6. The larger this intersection, relative to the size of the two core sets, the stronger the "force" pulling the two core nodes together. Hence the ratio of core set intersection size to core set union size captures the essential notion of inter-node similarity central to the strength/coupling criterion which underlies the entire graph decomposition problem.

2.2.3 Link Weights and the Core Set Definition.

The essential effect of link weights on the P'_{xy} measure can be summarized as follows. For a given pair of nodes (x,y) and corresponding core sets $\emptyset(x)$ and $\emptyset(y)$ (where it is assumed $|\emptyset(x) \cap \emptyset(y)| > 0$), the higher the weights on the links connecting nodes x and y to nodes in $\emptyset(x) \cap \emptyset(y)$, relative to the weights on the links within $\emptyset(x)$ and $\emptyset(y)$, the stronger the similarity between nodes x and y. Again,

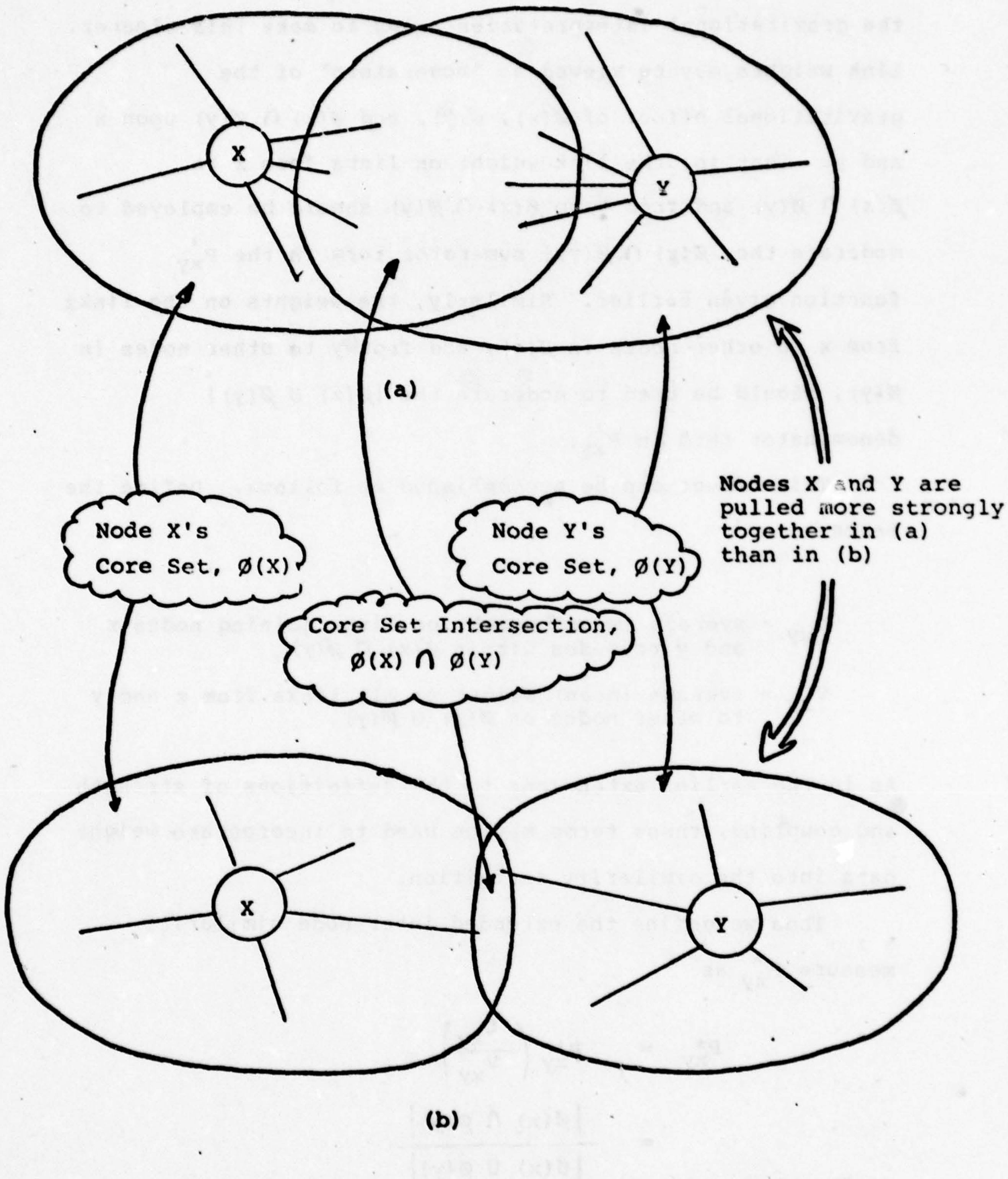


Figure 2.6

Core sets - the "gravitational" interpretation.

the gravitational interpretation helps to make this clearer. Link weights may be viewed as "moderators" of the gravitational effect of $\emptyset(x)$, $\emptyset(y)$, and $\emptyset(x) \cap \emptyset(y)$ upon x and y. That is, the link weight on links from x to $\emptyset(x) \cap \emptyset(y)$ and from y to $\emptyset(x) \cap \emptyset(y)$ should be employed to moderate the $|\emptyset(x) \cap \emptyset(y)|$ numerator term in the P'_{xy} function given earlier. Similarly, the weights on the links from x to other nodes in $\emptyset(x)$, and from y to other nodes in $\emptyset(y)$, should be used to moderate the $|\emptyset(x) \cup \emptyset(y)|$ denominator term in P'_{xy}.

This effect can be accomplished as follows. Define the terms:

U_{xy} = average (mean) weight on links joining nodes x and y to nodes within $\emptyset(x) \cap \emptyset(y)$

V_{xy} = average (mean) weight on all links from x and y to other nodes on $\emptyset(x) \cup \emptyset(y)$.

As in the earlier extensions to the definitions of strength and coupling, these terms may be used to incorporate weight data into the similarity definition.

Thus we define the extended inter-node similarity measure P^*_{xy} as

$$P^*_{xy} = P'_{xy}\left(\frac{U_{xy}}{V_{xy}}\right)$$

$$= \frac{|\emptyset(x) \cap \emptyset(y)|}{|\emptyset(x) \cup \emptyset(y)|}$$

As an example, consider the graph shown in Figure 2.7

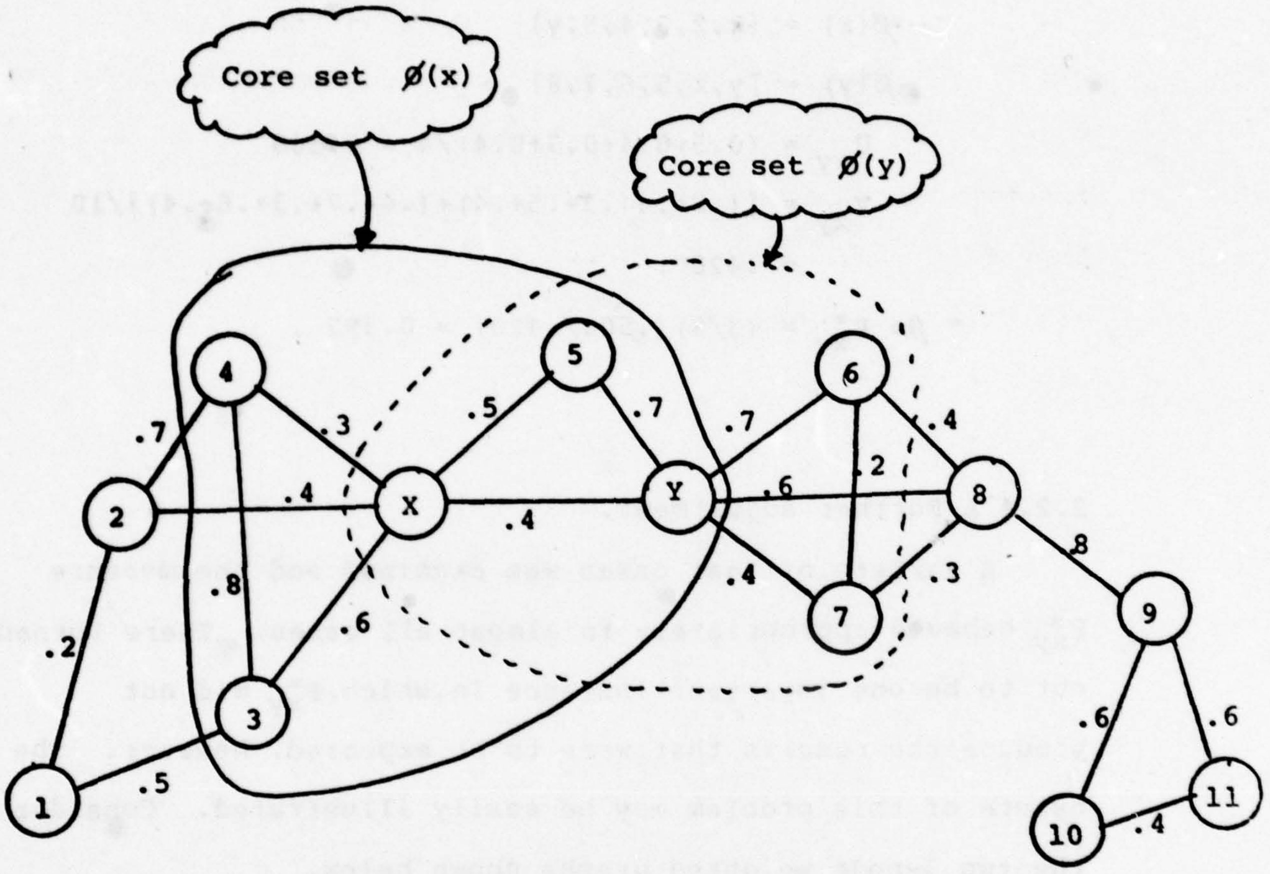


Figure 2.7

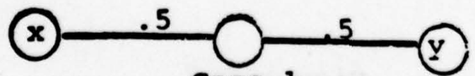
Core sets of a weighted graph.

(the same as the graph in Figure 2.5, with weights added to the links). As earlier,

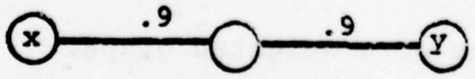
$$\emptyset(x) = \{x,2,3,4,5,y\}$$
$$\emptyset(y) = \{y,x,5,6,7,8\}$$
$$U_{xy} = (0.5+0.4+0.7+0.4)/4 = 0.500$$
$$V_{xy} = \{(.2+.4+.3+.5+.4)+(.4+.7+.3+.6+.4)\}/10 = .420 \;.$$
$$\text{So } P^*_{xy} = (3/9)(.500/.420) = 0.397 \;.$$

2.2.4 A Further Adjustment.

A variety of test cases was examined and the measure P^*_{xy} behaved appropriately in almost all cases. There turned out to be one important instance in which P^*_{xy} did not produce the results that were to be expected, however. The nature of this problem may be easily illustrated. Consider the two 3-node weighted graphs shown below.



Case 1



Case 2

Clearly, we would desire that the similarity between x and y in case 2 be greater than the same similarity in case 1. However, due to the nature of the function P^*_{xy}, it turns out that in both cases $U_{xy} = V_{xy}$, so the ratio of these terms cancels out of the expression. The result is that the x-y similarities are equal in both cases.

In general, this cancellation effect will occur whenever the following condition holds:

$$\emptyset(x) \cap \emptyset(y) = \emptyset(x) \cup \emptyset(y) - \{x,y\}.$$

That is, whenever the core set union and intersection differ only by the nodes x and y themselves. This is not at all an unusual occurrence, hence an additional modification must be made to P^*_{xy} in order to override this cancellation effect.

A good deal of experimentation with a number of posssible adjustments led to the following simple change: we replace the term U_{xy} with U^2_{xy} in the P^*_{xy} definition. That is, we define

$$P_{xy} = (U^2_{xy})P^*_{xy}$$

$$P_{xy} = \frac{|\emptyset(x) \cap \emptyset(y)|}{|\emptyset(x) \cup \emptyset(y)|} * \left(\frac{U^2_{xy}}{V_{xy}}\right)$$

This simple change really has the effect of scaling all the P^*_{xy} values by the factor U_{xy}. In the special case described above, it insures that the link weights on the x-y path do have an impact, as desired.

To see this impact, consider the two 3-node graphs discussed a moment ago. Now we would find that

$$\text{Case 1}: P_{xy} = (1/3)(0.5^2/0.5) = 0.167$$

$$\text{Case 2}: P_{xy} = (1/3)(0.9^2/0.9) = 0.300,$$

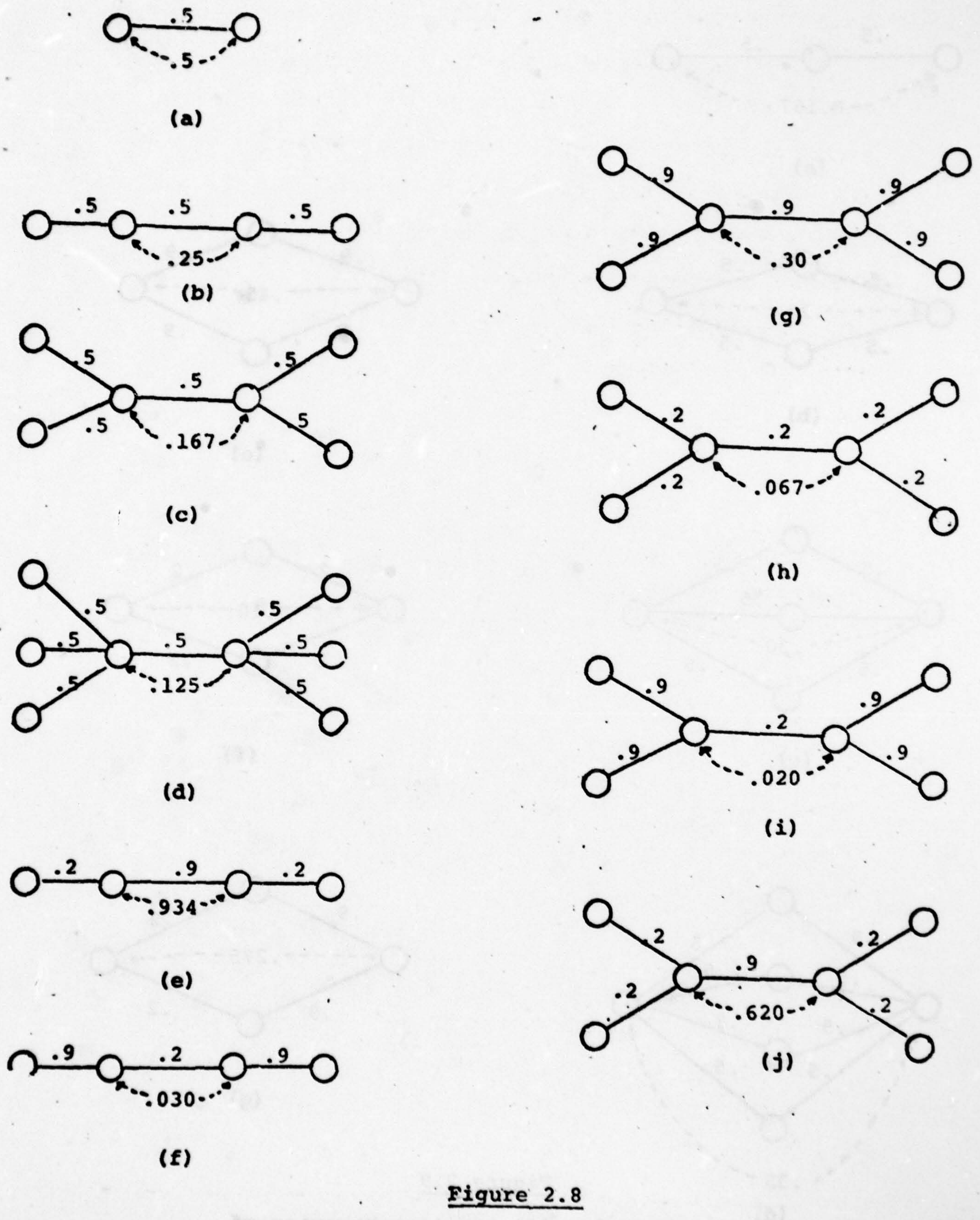
a much more reasonable result than that obtained earlier.

2.2.5 Some Test Cases Using P_{xy}.

In order to better see the operational effect of the similarity function P_{xy}, it is worthwhile examining sequences of simple graphs, in which a single factor is changed from one step to the next. Figures 2.8 and 2.9 contain some enlightening sequences. In Figure 2.8, in the sequence a-b-c-d, more and more nodes are added to $\emptyset(x)$ and $\emptyset(y)$ individually, but these nodes do not impact the intersection. All link weights are kept constant at 0.5. The effect is to "pull apart" nodes x and y, i.e., to make it increasingly desirable that two clusters should be created by severing the x-y link. As is shown, the similarity P_{xy} correspondingly decreases through the sequence, as is desired.

Now in the sequence b-e-f, the graph structure is held fixed while link weights are altered. In (b), $P_{xy} = 0.25$. In (e), the link weight W_{xy} is increased from 0.5 to 0.9, while the other weights are decreased from 0.5 to 0.2 (compare to (b)). We would expect the similarity P_{xy} to increase, and it does (to 0.33). In (f), the opposite changes to link weights result in P_{xy} decreasing (as expected) to 0.03.

Finally, in the sequence c-g-h-i-j, similar intuitively correct results are also seen. In particular, if (i) is compared to (f), it is seen that the impact on P_{xy} is slightly greater in the former case, when link weights are changed from all 0.5 to the 0.9-0.2-0.9 pattern. This is reasonable, since the extra nodes in (i) would be expected to exert an even greater "gravitational" effect on x and y than in (f) as a result of the link weight change.



<u>Figure 2.8</u>

Examples illustrating behavior of similarity measure.

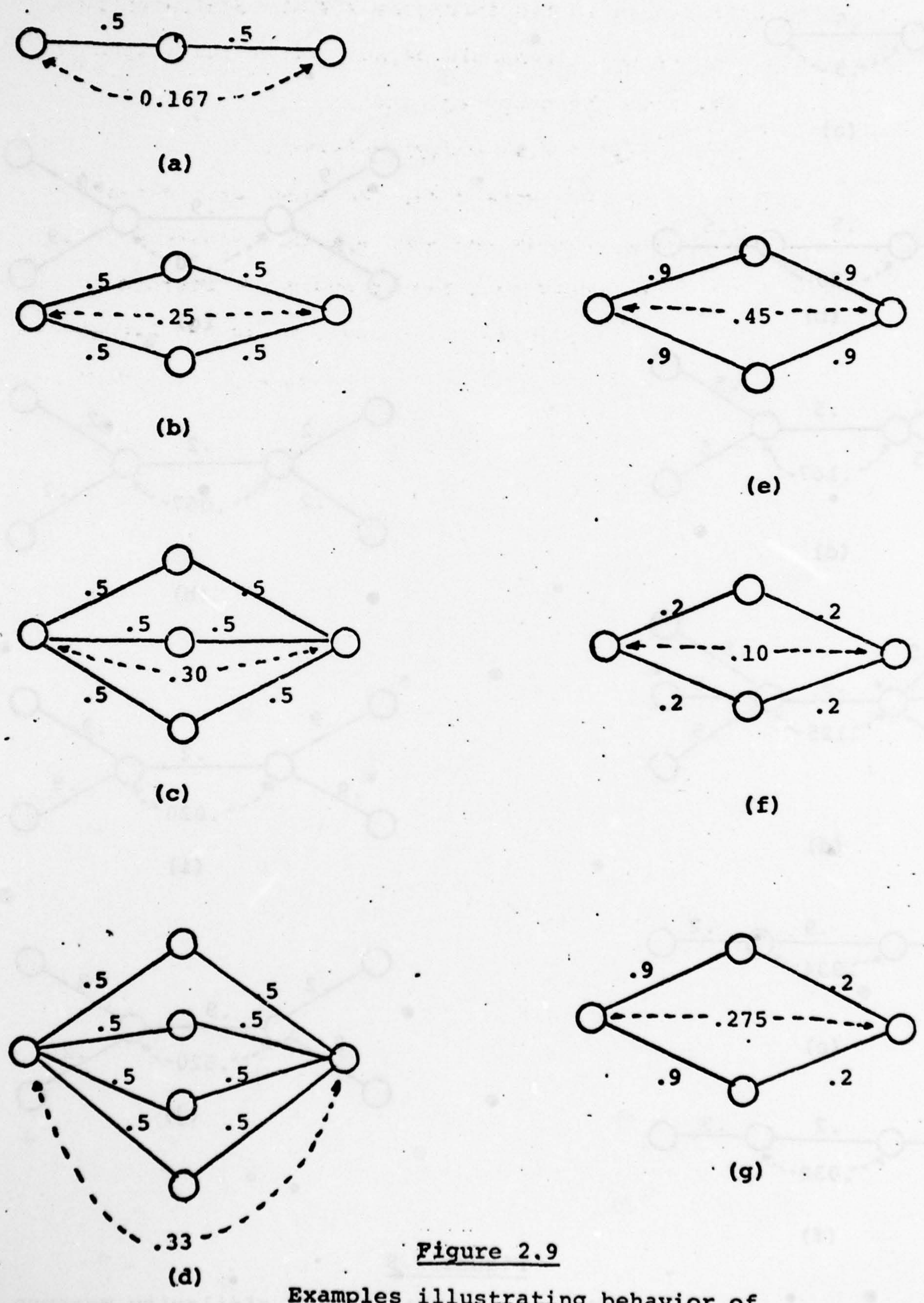


Figure 2.9

Examples illustrating behavior of similarity measure.

The patterns exhibited in Figure 2.9 are equivalently intuition-supporting. It should be noted that the graphs in Figure 2.9 all have the property that

$$\emptyset(x) \cap \emptyset(y) = \emptyset(x) \cup \emptyset(y) - \{x,y\} .$$

Hence if the measure P^*_{xy} were used, the sequence b-e-f-g would all exhibit precisely the same similarity between nodes x and y. Obviously this result would not conform to intuition: in (e), for instance, x and y would be expected to be more similar than in (f). The use of P_{xy} instead of P^*_{xy} insures that this is the case.

3 Clustering Analysis Techniques Using the Extended Model.

The main purpose of cluster analysis is to "group similar objects" (Hartigan 75). While there is a large number of individual techniques available in the clustering literature, they may be broadly categorized into two groups: agglomerative (or "bottom-up") techniques and partitioning ("top-down") techniques. The former class of techniques begins with each point (node) being viewed as a separate cluster, then proceeds to join together the "most similar" pair of clusters. The merging process is repeated until a single cluster remains.

Partitioning techniques move in the opposite direction. Beginning with a single encompassing cluster, they proceed to break up the cluster into two (or more) sub-clusters. After each cycle, a decision must be made as to which current cluster should be partitioned next.

There are also other "hybrid" techniques that possess aspects of both classes. "Leader" techniques, for example, begin by partitioning the entire set into a set of especially strongly connected clusters ("leader" clusters) plus unallocated elements. Special methods are then used to decide what to do with the unallocated points - i.e., assign them to one of the leader clusters, or group some of them together to form additional clusters.

A discussion of clustering techniques used in the earlier SDM analysis is given in (Andreu 78) and summarized in (Huff 79). In general, the most effective techniques, in terms of both algorithm execution speed and ability to locate good decompositions, have been basic bottom-up clustering approaches (to be described shortly). Andreu experimented with some other approaches but found them to be too inefficient in terms of solution time to be used on graphs of nontrivial size. A new top-down partitioning technique that exhibits reasonable efficiency has recently been developed by this author (Huff 79), and is compared in performance against the various clustering algorithms in Section 3.4.4. Many other techniques exist that have yet to be explored, and a few of the most promising ones are briefly discussed in a later section of this report.

3.1 Four Hierarchical Clustering Techniques.

At this point four different hierarchical clustering techniques that have been used most frequently in SDM analysis, and which are presently included in the SDM PL/1 analysis package, will be described. All four techniques are based on similarity (as opposed to distance) coefficients. Figure 3.1 illustrates three clusters of graph nodes, shown for simplicity as points. (Once a similarity matrix has been computed, the information originally conveyed by the graph links and weights has been absorbed into the similarity values, hence the actual graph structure loses importance.) Assume that the original unclustered points (nodes) have been partially clustered to the stage indicated in Figure 3.1. The next clustering decision is the determination of the best pair of clusters,

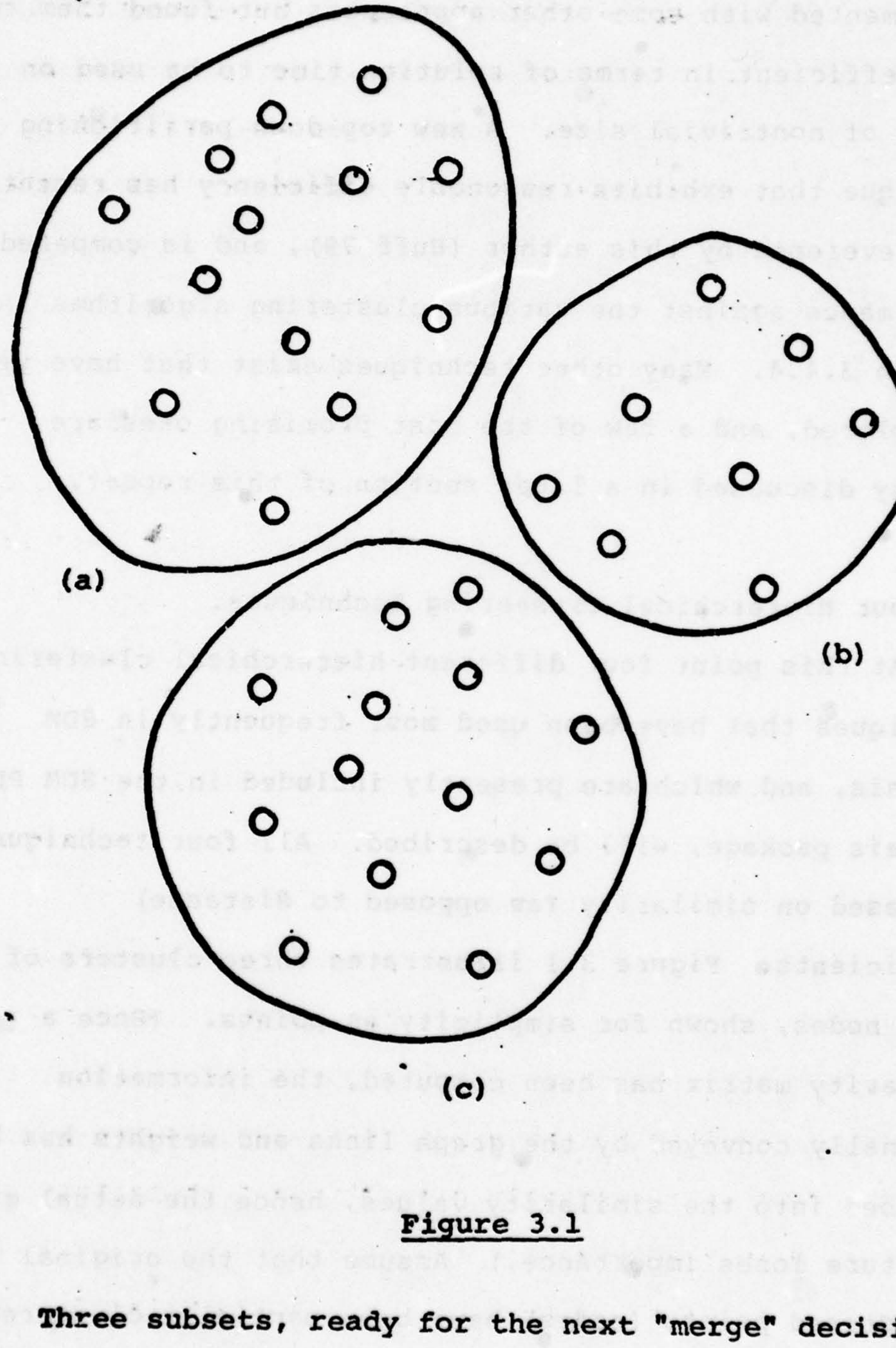


Figure 3.1

Three subsets, ready for the next "merge" decision.

(a,b), (a,c), or (b,c), to be merged at the next step.

For each of the methods (termed HIER1, HIER2, HIER3, and HIER4) a criterion value is calculated for each cluster pair that may be merged at that step. The cluster pair with the largest criterion value is then merged to form a single cluster, producing the next level up in the clustering hierarchy. This process is then repeated until a single cluster remains.

3.1.1 Single Linkage Clustering (HIER1).

The essence of the clustering decision involves the question of what is meant by the closeness between two sets of points, given that the closeness between each pair of points is quantitatively known. Perhaps the simplest interpretation of set closeness is that employed in the "single linkage" clustering algorithm: the closeness between two sets A and B is taken to be the closeness between the closest pair of points (a,b) such that $a \in A$ and $b \in B$. The algorithm derives its name from the fact that only a single pair of points (a,b) need be especially "close" in order that the entire sets A and B be judged to be "close". While the single linkage algorithm generally gives good results, it can sometimes lead to unusual clustering patterns, notably the "strung out" pattern illustrated in Figure 3.2.

Single linkage clustering, then, is formalized accordingly:

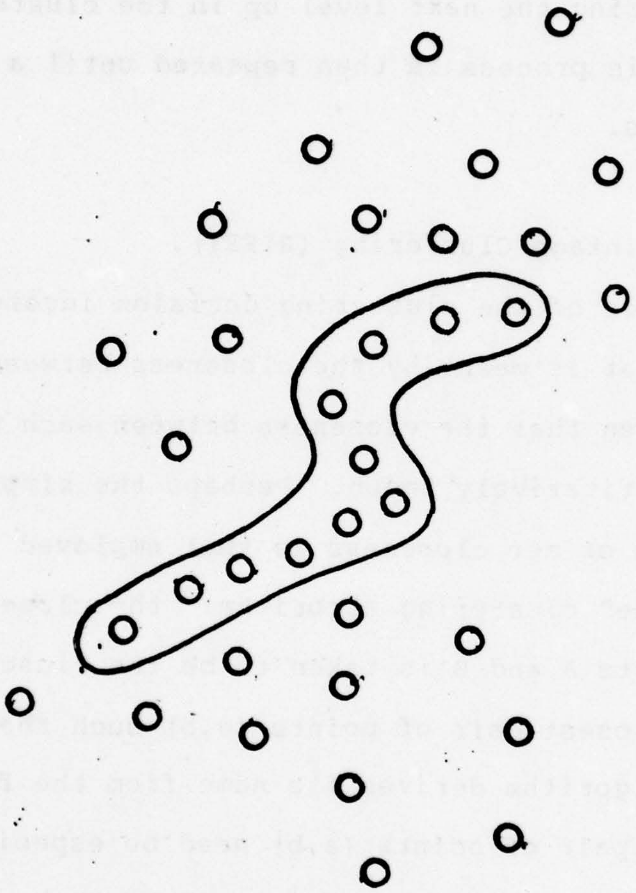


Figure 3.2

Single-linkage clustering anomaly.

For each pair of clusters (X,Y), calculate

$$P_{XY} = \{ \max_{\substack{i \varepsilon X \\ j \varepsilon Y}} (p_{ij}) \},$$

where p_{ij} = similarity between points i and j.

Then merge the clusters (X^*,Y^*) such that

$$P_{X^*Y^*} \geqslant P_{XY} \quad \forall \quad X,Y.$$

3.1.2 Complete Linkage Clustering (HIER2).

Single linkage may be viewed as a "risk-prone" algorithm: the algorithm is willing to presume that, if the point pair (a,b) is close, the other points in A and B will be close also, hence A and B should be merged. In contrast, complete linkage is a "risk-averse" algorithm. Rather than make the assumption stated above, this algorthm insures it by seeking to merge the cluster pair such that the least similar points are closest. That is, under complete linkage, the pair of clusters to be merged at any stage is determined as follows.

For each pair of clusters (X,Y), calculate

$$P_{XY} = \{ \min_{\substack{i \varepsilon X \\ j \varepsilon Y}} (p_{ij}) \}$$

Then merge the cluster pair (X^*,Y^*) such that $P_{X^*Y^*} \geqslant P_{XY} \quad \forall \quad X,Y.$

Thus, single linkage follows a "maxi-max" rule, while complete linkage is a "maxi-min" technique.

3.1.3 Largest Pre-merge Centroid (HIER3).

If sets X and Y contains points $\{x_1, x_2, \ldots x_{n_x}\}$, $\{y_1, y_2, \ldots y_{n_Y}\}$, respectively, then the "similarity centroid" between these sets is defined as

$$D_{xy} = \frac{1}{n_x n_Y} * \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} P_{x_i y_j}$$

where $P_{x_i y_i}$ = the similarity coefficient between points $x_i \varepsilon X$ and $y_i \varepsilon Y$.

D_{XY} is a measure of "average similarity" between the sets X and Y.

The largest pre-merge centroid algorithm makes use of the D_{XY} measure: at any stage in the clustering, the cluster pair (X,Y) with the largest value of D_{XY} is selected for the next merge. Formally:

For each pair of clusters (X,Y), calculate D_{XY}, then merge cluster pair (X^*,Y^*) such that

$D_{X*Y*} \geqslant D_{XY} \;\forall\; X,Y$.

3.1.4 Largest Post-Merge Centroid (HIER4).

If set X contains points $\{x_1, x_2, \ldots\ x_n\}$ then the "similarity centroid" of the set X is defined as:

$$D_x = \frac{1}{\frac{n_x(n_x-1)}{2}} * \sum_{i=1}^{n_x-1} \sum_{\substack{j=\\i+1}}^{n_x} P_{x_i x_j}$$

where $P_{x_i x_j}$ = the similarity measure between points $x_i \varepsilon X$ and $x_j \varepsilon X$.

D_X is a measure of the "internal similarity" of the set X. The measure D_X is used in the largest post-merge similarity algorithm: the cluster pair (X,Y) such that, when merged, exhibits the largest internal similarity value, is selected as the next pair for merging. Formally, we have,

For each pair of clusters (X,Y), let $Z \leftarrow (X,Y)$ and calculate D_Z. Then merge the cluster pair (X^*,Y^*) such that $D_{Z^*} \geqslant D_Z \; \forall \; Z$.

3.2 Comparative Analysis of Clustering Methods.

Each of the clustering algorithms described in the previous section makes good intuitive sense. There is no obvious a priori way of choosing among them - i.e., of determining which one would tend to produce the best results in a typical SDM graph analysis. For this reason, all four algorithms are included in the SDM analysis package, and a user of the package may apply whichever one he chooses, or all four.

However, it is worthwhile to explore somewhat the question of dominance: does one (or more) of the algorithms tend to produce consistently superior decompositions relative to the others? A related question concerns efficiency: are certain of the clustering algorithms significantly faster executing than others in general? Of course, these questions cannot be answered for all possible cases. However, the experiment reported here will at least give some clues.

For this experiment, a "random graph generator" was developed, a series of graphs generated, and each clustering method applied to each graph. The results are discussed below.

The random graph generator (written in PRIME 400 extended BASIC) functions as follows. The user is requested to supply the node count for the graph to be generated, as well as the mean (Mg) and variance (Vg) of the distribution of links-per-node for the target graph. In developing the generator, it was assumed that the number of links per node would follow a normal distribution.

The generator, for each graph node, then proceeds to select normal random numbers from N(Mg,Vg) (truncated at 0) to use in allocating links to each node. To determine the "recipient" node for each such link, the generator again draws a random number, from the uniform distribution U[1,nodecount]. Finally, the link weight is likewise randomly selected, from U[0,1], modified so as to produce the values 0.1, 0.2, ..., 0.9 with equal liklihood.

A series of 7 graphs was generated using the random graph generator, to be used in the comparative analysis. Furthermore, it was felt that "random" graphs, while providing unbiased cases, probably do not exhibit as much internal structure as would real-world cases. Therefore, a set of six non-random graphs, generated by hand to exhibit significant clustering structure, was also included in the analysis. The specifications of the various test graphs are given in Table 3.1.

The results of the experiment are given in Table 3.2. This exhibit shows the measure for the best achieved decomposition for each of the four decomposition methods. A summary of the information from Table 3.2 is contained in Table 3.3. There it may be seen that all the algorithms except HIER1 (single linkage) produced a "clear winner" in at least one test case (in particular, in at least one randomly generated test case). HIER1, even though not producing a clear winner, did show the second-highest rate of producing "ties for best." The net result is that the second, third, and fourth algorithms somewhat outperform the first, although all four algorithms perform the decomposition task reasonably effectively.

A simple effectiveness ranking may be produced by arbitrarily assigning a 4 for producing a "clear winner", 3 for "tied for best", etc. The ranking that results is given in Table 3.4. This ranking reiterates the fact that three of the four algorithms (HIER2, HIER3, and HIER4) are essentially equivalently powerful in determing good decompositions, while the fourth (HIER1) seems somewhat less effective.

The conclusion to be gained from this effectiveness test is that it is useful (and important) to have a variety

	Graph ID Number	Number of Nodes	Mean Links per Node	Variance of LPN	Total No. of links
randomly generated	1	40	2.5	1.0	61
	2	22	3.5	1.5	42
	3	22	3.5	1.5	37
	4	15	3.5	1.5	27
	5	15	4.0	1.2	31
	6	15	2.5	1.2	24
	7	10	3.5	1.5	14
non-random	8	22	3.3	1.6	37
	9	15	2.9	0.9	22
	10	10	2.6	0.5	13
	11	6	2.3	0.5	7
	12	6	2.3	0.5	7
	13	6	2.3	0.5	7

Table 3.1

Specifications for random and non-random graphs.

	Graph ID Number	Number of Nodes	M value for best located decomposition (number of clusters in best result)			
			HIER1	HIER2	HIER3	HIER4
randomly generated graphs	1	40	.076 (5)	.098 (5)	.036 (3)	.021 (2)
	2	22	.04 (1)	.04 (1)	.10 (2)	.07 (3)
	3	22	.035 (3)	.046 (5)	.079 (4)	.080 (3)
	4	15	.24 (3)	.24 (3)	.24 (3)	.15 (4)
	5	15	.08 (1)	.11 (2)	.08 (1)	.08 (1)
	6	15	.052 (1)	.052 (1)	.052 (1)	.061 (2)
	7	10	.049 (2)	.056 (3)	.056 (3)	.085 (2)
non-random	8	22	.25 (3)	.29 (3)	.41 (4)	.41 (4)
	9	15	.39 (3)	.27 (2)	.39 (3)	.26 (3)
	10	10	.48 (3)	.48 (3)	.48 (3)	.48 (3)
	11	6	.28 (2)	.28 (2)	.28 (2)	.28 (2)
	12	6	.05 (1)	.05 (1)	.05 (1)	.05 (1)
	13	6	.06 (1)	.06 (1)	.06 (1)	.19 (2)

Table 3.2

Relative performance of the four hierarchical clustering techniques.

	HIER1	HIER2	HIER3	HIER4
Clear winner	0	2	1	4
Tied for best	5	4	6	4
2nd or tied for 2nd	4	5	5	3
3rd or tied for 3rd	3	2	1	1
4th or tied for 4th	1	0	0	1

Table 3.3

Relative performance for the clustering routines.

Weight	Category
4	Clear Winner
3	Tied for 1st
2	2nd or tied for 2nd
1	3rd or tied for 3rd
0	4th or ties for 4th

Algorithm	Ranking
HIER1	26
HIER2	32
HIER3	33
HIER4	35

Table 3.4

Ranking for clustering routines.

of algorithms at hand to bring to bear on such decomposition tasks. There is a nontrivial risk that, in amy given problem, any one algorithm will produce a considerably inferior graph decomposition.

All the clustering algorithms are reasonably fast in terms of computer execution time. Rough measurements obtained during the foregoing tests indicate that HIER1 is somewhat (e.g., 40 percent) faster than HIER2, HIER3, and HIER4. The latter three all seem to execute with roughly equal speed. As a benchmark, each of the latter three algorithms required approximately 3 CPU seconds on a 370/168 to perform complete clustering on the 40-node graph used in the comparative analysis. All the clustering algorithms are bounded by an execution speed proportional to n^2.

3.3 A "Greedy" Clustering Algorithm.

A somewhat different approach to clustering, motivated by the work of Ward (Ward 1963), was also investigated. Ward suggested that a general approach to clustering might be based on the notion of seeking to optimize some objective function to be specified by the investigator. Ward's own approach was to maximize the mean squared error function; that is, at each clustering step, the cluster pair to be merged is the pair that leads to the minimum increase (or maximum decrease) in the within-group mean squared error. Ward's particular criterion is only applicable to clustering problems wherein corresponding to each point is a vector of data values (e.g., the points might be individuals, the data values might be height, weight, etc.).

Ward's general objective function maximization approach may be applied to the SDM graph clustering problem, by adopting a different objective function. The most appropriate candidate for objective function is the decomposition "goodness" index, M. Following this criterion, at any stage in the clustering, the cluster pair (p,q) would be merged if the resulting impact on M was greater than for all other potential cluster mergers. More precisely:

At stage k, there are t clusters,

$\{C_{k_1}, C_{k_2}, \ldots, C_{k_t}\}$

Let $D_k = \{C_{k_1}, C_{k_2}, \ldots C_{k_t}\}$

$D_k(i,j) = \{C_{k_1}, C_{k_2}, \ldots \{C_{k_i} \cup C_{k_j}\}, \ldots C_{k_t}\}$

$= \{D_k - C_{k_i} - C_{k_j}\} \cup \{C_{k_i} \cup C_{k_j}\}$

= the new decomposition obtained from D_k by merging clusters k_i and k_j.

Define $\Delta M_k(i,j) = M(D_k(i,j)) - M(D_k)$.

Then merge clusters k_{i*} and k_{j*} such that

$\Delta M_k(i^*,j^*) = \max_{i,j} \{\Delta M_k(i,j)\}$.

Since this approach to clustering follows a path of local steepest ascent, it is conventionally termed a "greedy" algorithm: the algorithm tries to "get all it can," for the given objective function, at each step. Of course, such an algorithm is also "myopic," in that it only concerns itself with the best move at each stage; large increases early in the clustering may lead to poor results later on. It is not clear at this point whether such a greedy algorithm would do as well as the more conventional techniques discussed earlier.

In order to study the effectiveness of the greedy approach, this algorithm was programmed in PL/1 and added to the interactive analysis package. Certain of the graphs used in the comparative analysis reported in the previous section were also decomposed using the greedy algorithm.

First of all, the "greedy" algorithm in its present form is completely impractical, as it is simply too inefficient. Decomposing a 10-node graph, for example, requires on the order of 15 370/168 CPU seconds. This is not to say that this algorithm need necessarily be hopelessly slow, however, as discussed below.

Putting aside efficiency considerations momentarily, the "greedy" algorithm seems to perform in an unusual manner. In two of the test cases studied (cases 6 and 7), this algorithm produced the same decomposition as the best of the other four techniques. However, in two other cases (cases 10 and 11), "greedy" was unable to find the best decomposition, even though all four of the other algorithms did find it. While the cases studied here are limited because of the above mentioned efficiency problems, "greedy's" performance seems decidedly mixed.

The efficiency problem with "greedy" stems from the fact that a large number of calculations of M (the decomposition objective function) must be made, especially during the earlier clustering stages. In its present form, these M-calculations are carried out in their entirety (i.e., no approximations are introduced). While any given M-calculation is not terribly time-consuming (requiring perhaps .02 CPU seconds), the large number of such calculations made by "greedy" rapidly add up. For example, in the first clustering stage alone, for a 15-node graph, there would be

$$(14 + 13 + 12 + \ldots + 1) = 7(15) = 105$$

such calculations, requiring approximately two CPU seconds.

The "greedy" algorithm could probably be made acceptably efficient by developing an approximation to the M criterion that is employed within it. Such an approximation, potentially suitable to this algorithm, has been developed on behalf of yet another graph decomposition technique entirely, the interchange algorithm (see (Huff 79)). However, in light of the mixed performance of "greedy" in the early studies, as discussed above, its applicability and use within this algorithm has not yet been explored.

3.4 Other Approaches to Graph Decomposition.

In this section, a variety of other approaches to the decomposition of weighted graphs will be identified and briefly discussed. While the techniques to be presented here (with one exception) have not been incorporated into the SDM analysis package, they are all potentially appropriate for that purpose, pending further testing.

Future extensions to the package might include one or more of these techniques.

3.4.1 A Leader Technique.

The basic idea underlying leader (sometimes called "core") techniques is to isolate a few non-overlapping, strongly coherent subgraphs, then to perform additional analysis to determine what to do with the leftover nodes (if any).

Following the thinking underlying the goodness measure M, we would like to identify leader subgraphs that have especially high strength, and which are relatively weakly coupled to the other subgraph nodes (notably, to nodes in the other leader subgraphs).

Andreu derived such an algorithm for identifying leader subgraphs, described in (Andreu 78, pages 124-133). Andreu's algorithm is based on the notion of node connectivity. The connectivity of node i is simply the number of nodes that are linked to node i (recall Andreu worked with binary links). The algorithm then

(a) isolates a subset, U, of nodes with highest connectivity;

(b) finds the node of this subset with the largest "kernel subset," where the kernel subset of node i is the set of all nodes that are members of CS_i (the core set of node i) but not members of the core sets of other nodes in the subset U;

(c) isolates that kernel subset as a leader subgraph, and reduces the original graph accordingly;

(d) repeats the first three steps for the remaining nodes in U until one of the possible stopping conditions is reached.

This leader subgraph technique could be extended to analyze graphs with weighted links by re-defining the concept of connectivity. The following definition of c_i, the connectivity of node i, would serve to identify those nodes that are both thickly (many links) and strongly (high link weights) connected in the network:

Let CS_i = core set of node i

Then c_i = connectivity of node i

$$= \sum_{\substack{j \varepsilon CS_i \\ j \neq i}} a_{ij}$$

That is, c_i is simply the sum of the weights on the links within CS_i.

Re-defining c_i in this fashion, then applying the remaining steps of Andreu's algorithm, will tend to isolate leader subgraphs which are both thickly and strongly connected within themselves. Both characteristics are important for good leader subgraphs, as it is important, in decomposing the requirements graph, to both avoid cutting a large number of links and avoid cutting links with high weights.

One potential difficulty with the leader subgraph technique, one which Andreu does not really address, concerns what to do with leftover nodes (i.e., nodes that do not become members of one of the kernel subsets). There are various approaches that may be taken - lump leftovers into the subgraph to which each is most strongly connected, group certain leftovers to form a new cluster, etc. - but specific implementation techniques may prove challenging.

3.4.2 The Bond Energy Approach.

A new cluster analysis algorithm was developed by McCormack (McCormack, et. al., 71), and has recently been applied effectively to certain cluster applications similar in nature to SDM decomposition (Hoffer & Sevarance 75). This technique operates directly upon a graph adjacency matrix. By permuting the rows and columns of the matrix in such a way as to push numerically larger array elements together, this algorithm serves to identify the natural groups and clusters that occur in the data, as well as the associations of these groups with one another. The authors have named their algorithm the "bond energy" technique. Central to it is a "measure of effectiveness," or ME, which is used to quantify the "clumpiness" of a given permutation of the rows and columns of the array. The ME is larger for an array which possesses dense clumps of numerically large elements as compared to an equivalent array in which the rows and columns have been permuted so that the large elements are more uniformly distributed.

McCormick suggests the following measure of effectiness:

$$ME = \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})$$

Essentially, this ME is the sum over all node pair of the "bond energy" for each node pair, where bond energy for nodes (i,j) is calculated as

$$a_{ij} (a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})$$

In matrix terms, the bond energy for node pair (i,j) is the product of the four nearest-neighbor adjacency values (weights) for that node pair.

The algorithm used to locate the row/column permutation with the highest ME involves reducing the overall total $(N!)^2$ permutations to 2N calculations by applying the nearest-neighbor feature of the maximand. While the algorithm does not guarantee optimal ME, its use has shown it to produce very good results in general. This algorithm basically involves arbitrarily placing one column in a permuted position, then placing each remaining column in turn in the position that produces the greatest contribution to ME. In the general case, this procedure would have to be repeated on the matrix rows; however, in the case of a symmetric matrix (the present case), no such row permutations need be executed.

The bond energy algorithm (BEA) suffers from the disadvantage that it gives no hierarchical trace. That is, it produces a single best clustering, not a hierarchical sequence of clusters. Consequently, additional mechanisms would have to be added "on top" of the BEA itself to allow such exploratory marginal analysis. Furthermore, some of the clustering and partitioning algorithms discussed earlier might be effectively combined with the BEA to allow such marginal analysis to be performed easily.

The BEA was illustrated by McCormack in an example problem that bore much similarity to the usual SDM context. This suggests that it might be a particularly fruitful avenue for investigation for SDM analysis.

3.4.3 Node Tearing Techniques.

The central idea underlying this class of decomposition techniques is to locate small separating sets - i.e., sets of nodes with low cardinality such that their removal from the graph splits the graph into two unconnected subgraphs. In this sense, the network is said to be "torn" in half - hence, node tearing.

A new algorithm for node tearing was reported recently by Sangiovanni-Vincentelli (Sangiovanni-Vincentelli, et. al., 77), in the context of electrical circuit design (a context surprisingly similar to that of SDM). Many of the basic concepts used in this technique are similar to those used by Andreu in the leader subgraph approach discussed earlier (core set, connectivity, etc.), and their algorithm has much the same flavor as that of Andreu's, although its objective is somewhat different.

In particular, the algorithm proposed by Sangiovanni-Vincentelli presumes a binary graph. As in the case of the leader algorithm, however, it appears that it is feasible to extend it to incorporate link weight information into its operation. The key decision point in the algorithm is the choice of the "next iterating node" (see the reference for details). Sangiovanni-Vincentelli suggests a "greedy" strategy for making this decision, namely, to choose the node that minimizes the connectivity of a certain subgraph. If this criterion were changed, to "minimize the sum of the weights on the links within that subgraph," the modified algorithm should function properly and take account of link weights appropriately. In particular, in the case where link weights are all equal, this modified algorithm should lead to the same results as the original.

3.4.4 The Interchange Algorithm.

Another top-down hierarchical partitioning technique has been developed, by the present author, specifically for the SDM graph decomposition problem. This algorithm successively partitions the "current graph" into two subgraphs, using a criterion derived directly from the decomposition goodness measure, M. Its detailed operation, and examples of its use, are given in (Huff 79), and will not be further elaborated upon here.

It is enlightening to compare the effectiveness of the interchange algorithm against the four hierarchical clustering algorithms studied earlier. Each of the graphs used in the comparative analysis of the clustering algorithms (Section 3.2) was also decomposed using the interchange algorithm. The results are summarized in Table 3.5. The net result is that in all but one case the interchange algorithm produced at least as good a decomposition as the best clustering routine, and in five of the 13 cases produced a better result than the best clustering routine. In the one case where interchange failed to produce a result as good as the best clustering routine, it found one almost as good.

Weighed against this superior performance is the fact that interchange may be more costly to use in terms of computer (and, to some extent, human) resources. For instance, interchange required approximately 9 CPU seconds to decompose the 40-node graph (case 1 in Table 3.2), whereas the clustering routines each required no more than about 3 seconds. Also, the software used to execute interchange may be used in an "exploratory" fashion (the main parameters being subgraph size and choice of next subgraph for partitioning). In such a usage pattern, the user of the package would have to spend somewhat more time using interchange to reach an optimal result for a given graph. Alternately, an automatic "governor" may be called to execute interchange according to a predefined pattern of steps (described in detail in (Huff 79)). This approach would save the user time, but remove the possibility of exploring for superior decompositions.

	Objective function (M) values	
Graph ID number	Best clustering result (algorithm no.)	Interchange result
1	0.098 (2)	0.123 *
2	0.10 (3)	0.157 *
3	0.08 (4)	0.107 *
4	0.24 (1,2,3)	0.24 =
5	0.11 (2)	0.125 *
6	0.061 (4)	0.074 *
7	0.085 (4)	0.075 +
8	0.41 (3,4)	0.94 *
9	0.39 (1,3)	0.39 =
10	0.48 (1,2,3,4)	0.48 =
11	0.28 (1,2,3,4)	0.28 =
12	0.05 (1,2,3,4)	0.05 =
13	0.19 (4)	0.19 =

* cases where interchange exceeded best clustering algorithm;
+ case where interchange failed to co as well as best clustering algorithm;
= cases where interchange and best clusteing algorithm did equally well.

Table 3.5

Comparison of interchange and best result obtained using hierarchical clustering.

Weight	Category
5	Clear winner
4	Tied for first
3	Second or tied for second
2	Third or tied for third
1	Fourth or tied for fourth
0	Fifth or tied for fifth

	Algorithm				
	HIER1	HIER2	HIER3	HIER4	INTERCHANGE
Clear winner	0	0	0	1	6
Tied for first	5	4	5	4	6
2nd or tied	1	4	3	4	1
3rd or tied	3	4	4	3	0
4th or tied	3	1	1	0	0
5th	1	0	0	1	0
Composite score	32	37	38	39	57

Table 3.6

Comparison of the weighted performance of interchange and the four hierarchical clustering routines.

If we re-do the weighted comparison, using a weight of 5 for a "clear winner," 4 for a "tied for first," etc., in a fashion similar to that given in Table 3.4 earlier, the new results (with interchange included) are as shown in Table 3.6. Interchange earns a score of 57, while the clustering techniques score from 32 to 39. All in all, then, the interchange algorithm is seen to be a powerful, if somewhat less efficient, technique for SDM graph decomposition. The interchange algorithm has been incorporated into the current SDM analysis package, and its use is described briefly in Appexdix A in the context of documentation of the SDM analysis package.

4 A Case Study Using Interdependency Weight Extensions.

In order to illustrate the application of the various techniques discussed in this paper, including the use of the SDM analysis package, we present here a decomposition analysis of a particular small design problem. The problem addressed is one that has been used in this research effort in the past: a set of 22 requirements and interdependencies for the design of a database management system. The requirements were originally developed as a simple test vehicle in an early phase of the SDM research (see Andreu 78), and have been referred to on a number of occassions. Specifically, this investigator made use of the 22-node system in illustrating various potential extensions to the SDM representational model (Huff & Madnick 78).

Source statements of the system's requirements, and the interdependency relationships, are given in the foregoing reference. Figure 4.1 shows the graphical representation for this design problem. The interdependency weights are given in coded terms, as discussed earlier: W means "weak," A, "average," and S "strong." A similar scheme is used to label strengths of interdependency relationships.

In the earlier analysis of this 22-node design problem, the basic (binary) graph model was used. An analysis of that graph produced the following best decomposition:

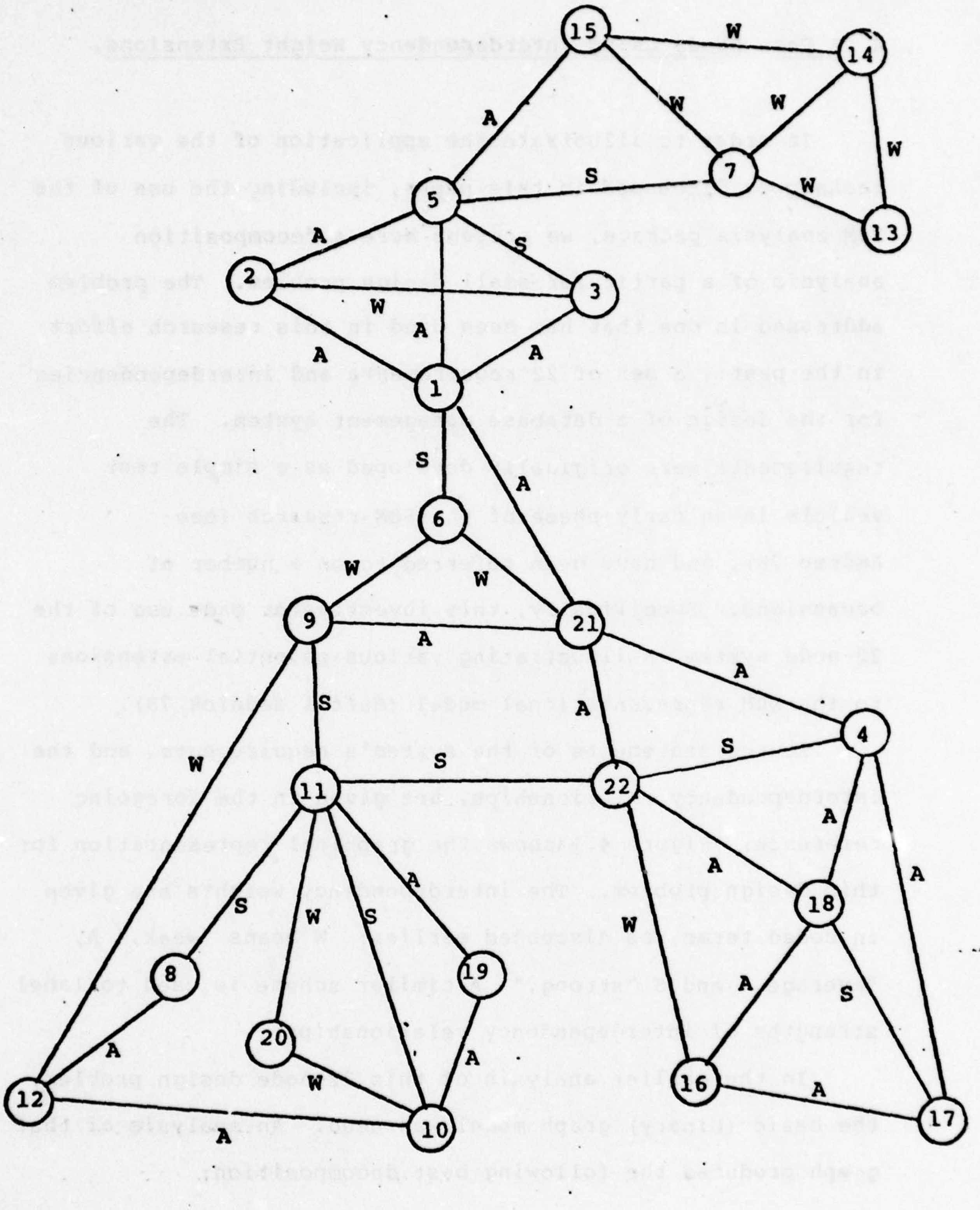


Figure 4.1

The 22-node DBMS requirements graph.

CLUSTER	NODES INCLUDED
1	1,2,3,5
2	6,9,21
3	4,16,17,18,22
4	8,10,11,12,19,20
5	7,13,14,15 .

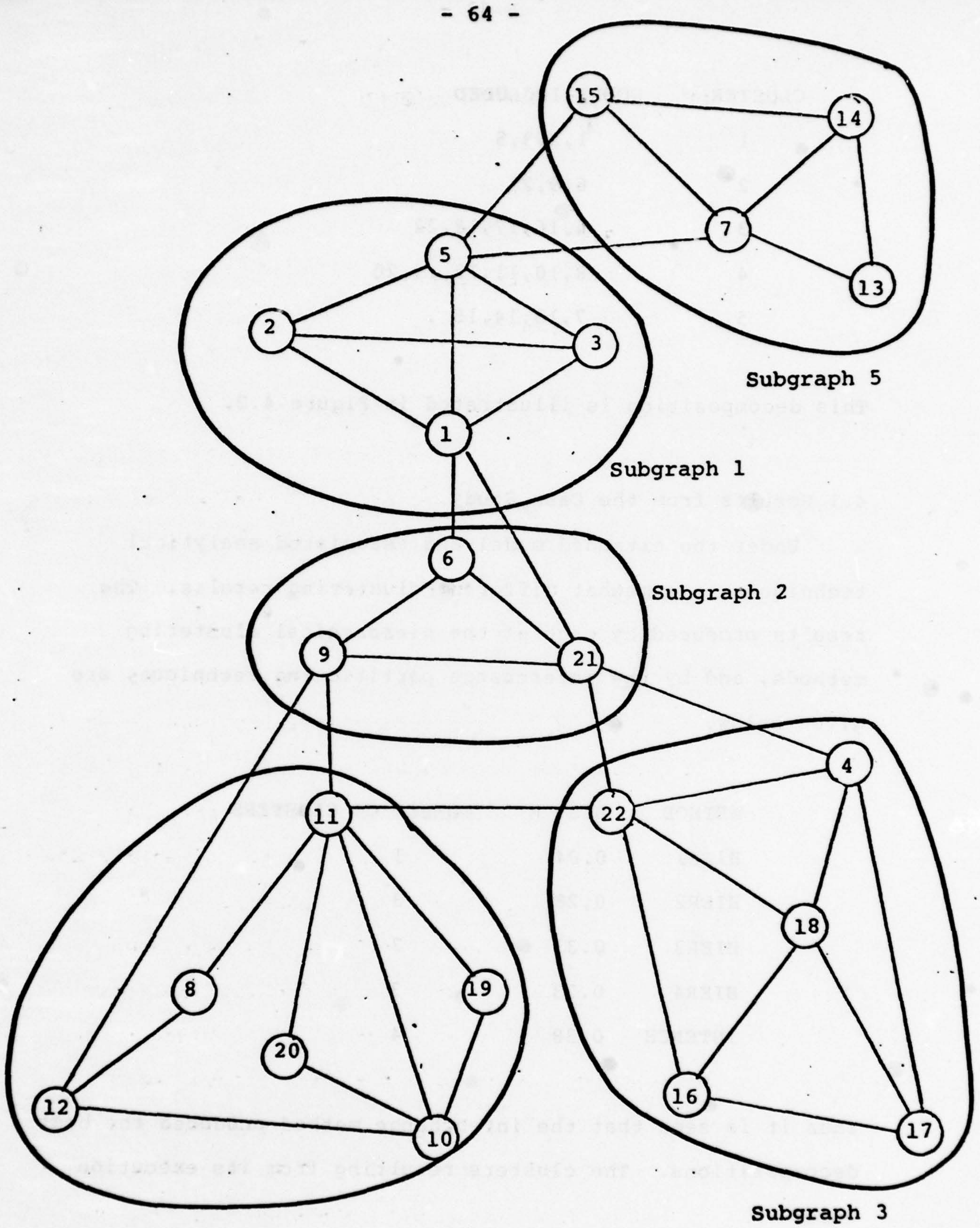
This decomposition is illustrated in Figure 4.2.

4.1 Results from the Case Study.

Under the extended model and associated analytical techniques, a somewhat different clustering results. The results produced by each of the hierarchical clustering methods, and by the interchange partitioning technique, are given below.

METHOD	BEST M	NUMBER OF CLUSTERS
HIER1	0.04	1
HIER2	0.28	3
HIER3	0.33	3
HIER4	0.23	3
INTERCH	0.38	4

Thus it is seen that the interchange method produced the best decompositions. The clusters resulting from its execution were:



<u>Figure 4.2</u>

Best decomposition of the unweighted 22-node graph (Andreu 78).

CLUSTER	NODES TO BE INCLUDED
1	1,2,3,5,6
2	4,16,17,18,21,22
3	8,9,10,11,12,19,20
4	7,13,14,15

This clustering is illustrated in Figure 4.3.

The difference between the two results resides entirely in the treatment of nodes 6, 9, and 21. In the earlier case, these three nodes were lumped together to form their own cluster. However, in the present analysis, each node has been moved into one of the other clusters. Comparison between Figures 4.2 and 4.3 indicates the reason for the change: the difference resides in the relative link weights on the links extending from nodes 6, 9, and 21. It may be seen that, by partitioning in the manner illustrated in Figure 4.3, two links of "W" weight and two of "A" weight are cut. With the partition of Figure 4.2, however, two "S" links, three "A" links, and one "W" link would have been cut. Clearly, the latter partitioning is more "costly" in terms of link cuts. If the strength of subgraph {6,9,21} was quite high, this more costly set of link cuts might have "paid off"; however, as the links (6,9), (6,21), and (9,21) are not particularly strong (having weights of W, W, and A respectively), such is not the case. Consequently, the new decomposition produced by the extended analysis techniques can in this case be reasonably justified as being an

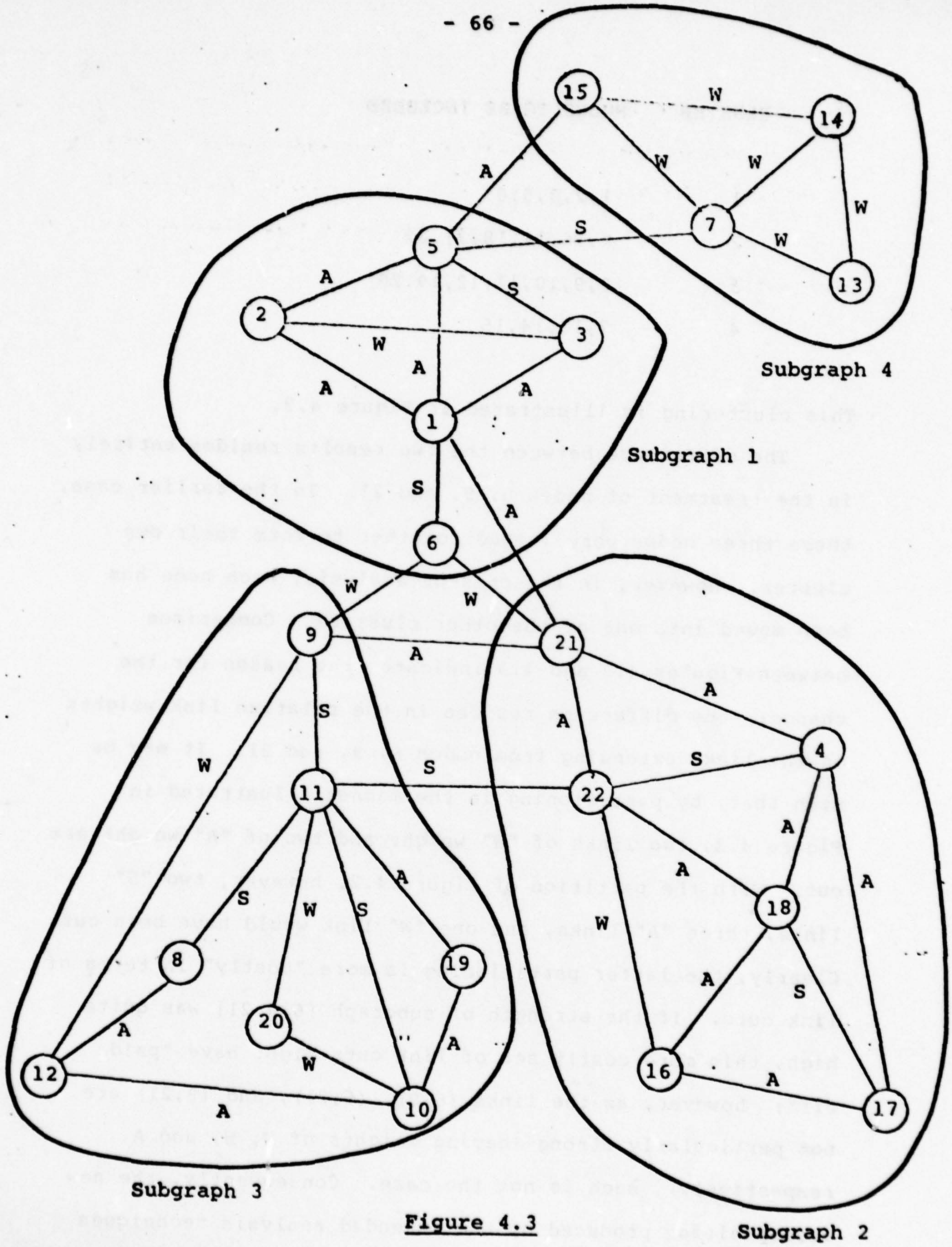


Figure 4.3

Best decomposiiton of the weighted 22-node graph.

improvement over the earlier approach.

As part of the documentation of the SDM analysis package included in the appendix, the terminal session that produced the foregoing results for the 22-node graph is included as Appendix B.

5 SDM Analysis Using Other Model Extensions.

To this point we have addressed in some detail the question of how to extend the key analytical mechanisms used in performing SDM decomposition analysis to incorporate interdependency strength factors. However, interdependency strength is not the only extension to the SDM representational model proposed in (Huff & Madnick 78). Other proposed extensions include

a) interdependency similarity relationships and accompanying strength factors;

b) implication relationships between requirements and between interdependencies;

c) hierarchical implication relationships.

That report made it clear that the various kinds of model extensions that were proposed there were not necessarily appropriate bases for extensions to the full set of decomposition techniques. To take a case in point, it may be clear that directed (implication) relationships between requirements exist, are relatively easily identified by systems analysts, and may be appropriately represented in the SDM model. It may not, however, be at all obvious how the information contained in such relationships ought to be used to affect good decomposiitons of the requirements graph.

Study of the kinds of model extensions identified in (Huff & Madnick 78) has suggested that some of the proposed extensions may be more generally relevant than others. In particular, application of all the proposed extensions to a particular case study, the 22-requirement DBMS discussed in the previous section, indicated that, of all of the proposed secondary extensions, the one most frequently and usefully applied was interdependency similarity relationships. The DBMS example discussed there gave rise to ten such relationships, whereas only three inter-requirement implication relationships could be identified, and only one hierarchical implication relationship identified.

On the basis of this example, and of the broader insight gained in studying it and other similar sets of requirements, it may be tentatively concluded that, if the SDM analysis techniques are to incorporate any of these "secondary" extensions to the main weighted-graph techniques reported earlier in this paper, then attention should be directed toward interdependency similarity relationships first. The purpose of this section is to explore possible ways in which this may be accomplished.

5.1 Interdependency Similarity Relationships.

To briefly review the nature of interdependency similarity relationships (hereafter termed "ISR's"), the following is quoted from (Huff & Madnick 78):

> "Two or more links (interdependencies) may represent the same, or closely related, implementation issues. A simple example of this possibility is illustrated in Figure 5.1. The links joining requirements 1 and 2, and 2 and 3, both represent the interdependency "ISAM organization," an implementation consideration through which both requirement pairs (1,2) and (2,3) are deemed by the designer to be interdependent.
>
> In this example, the two links represent entirely the same implementation issue. In

> general, the degree of "sameness" between two or more implementation issues will generally be less than 100 percent in the eyes of the designer, due to the inherent fuzziness in the specification of both functional requirements and implementation schemes. The judgment as to whether a given pair of links "really" represent the same implementation issue is, again, a designer decision.
>
> Going one step further, a weight factor could be associated with the similarity assessment to represent the extent to which the designer judges the two implementation issues to be the same. That is, such a weight would correspond to the _extent of overlap_ between the implementation issues, in the designer's estimation."

The question at this point is not what ISR's are, not how they ought to be logically represented or viewed, but rather how they may be incorporated into the decomposition analysis. Two different approaches present themselves, both of which have been investigated.

5.1.1 Modification of Similarity Coefficients.

The main effect of ISR's is related to the effect of interdependencies themselves (essentially, an ISR may be viewed an an "interdependency between two interdependencies"; alternately, they may be likened to Chen's concept of "relationship relations" (see Chen 77)). Whereas the importance of interdependencies is to suggest that the associated requirements be grouped together in a decomposition, the proper interpretation of an ISR is to suggest that the associated _interdependencies_ are related and ought to be grouped together. However, since interdependencies are not "things," this statement has to be taken to mean that the system requirements which correspond to the related interdependencies ought to be grouped together.

The foregoing is actually harder to say than illustrate. Figure 5.2 indicates that interdependencies (1,2) and (2,3) are related. A good decomposition algorithm ought, therefore, to operate so as to group (1,2) and (2,3) into a common subgraph - i.e., to group requirements 1, 2, and 3 together. This is not to say that such a grouping <u>must</u> occur, of course, only that our preference for such a decomposition would be stronger than would be the case were there ISR no so assessed.

One approach to adjusting the decomposition algorithms to take account of this issue would be to modify the particular similarity coefficients associated with the affected requirements. In the foregoing example, the similarity coefficients (p_{12}, p_{13}, p_{23}, p_{21}, p_{31}, p_{32}) would all need to be modified - i.e., increased - so as to reflect our judgment of the extent to which the ISR makes these three requirements "more similar" to each other than they otherwise would be.

As usual, there is no objective rule to be followed, and we must be guided again by our intuition. However, some considerations to be kept in mind include:

a) however much the increase in the particular coefficients, they should not be increased beyond 1.0, the accepted upper limit for p_{ij};

b) a weight factor may be attached to the ISR, in a manner parallel to that for interdependency strengths, and may be used in the adjustment of the similarity coefficients.

A reasonable similarity modification technique, which

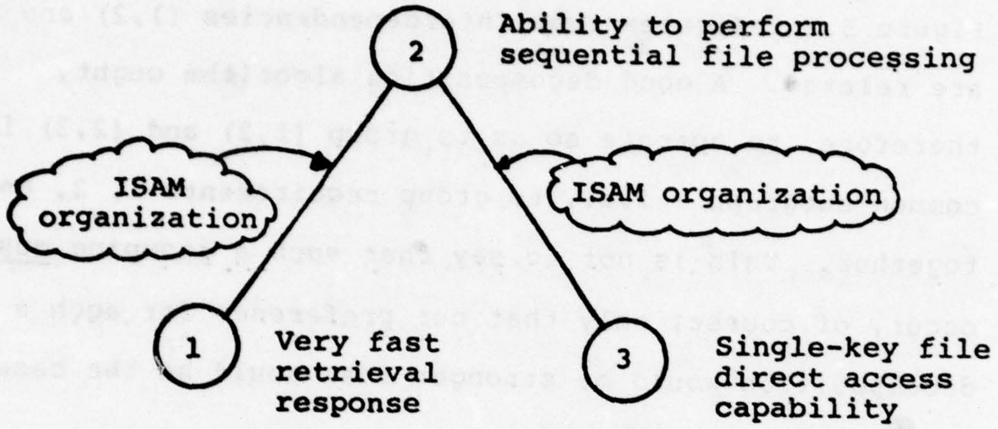

2
Ability to perform sequential file processing
ISAM organization
ISAM organization
1
Very fast retrieval response
3
Single-key file direct access capability

Figure 5.1

A simple interdependency similarity relationship (ISR)

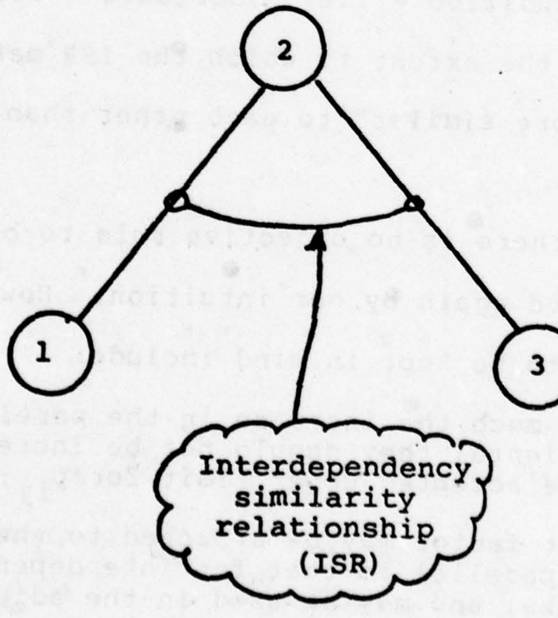


Figure 5.2

Representation of the ISR of Figure 5.1

observes the two conditions given above, may be stated as follows. Define the modified similarity between the affected requirements to be

$$p'_{ij} = \min \{ 1.0, p_{ij}*(1.0+v) \},$$

where i and j range over the three (or possibly four) affected node pairs, and v is the weight on the associated ISR.

For instance, if v were taken to be 0.5, the suggested modification would increase each affected similarity coefficient by 50 percent, to a maximum of 1.0.

This similarity modification approach has been incorporated, for testing purposes, into the SDM analysis package. Its efffectiveness as compared to a different approach (to be discussedd momentarily) will be reported below.

One serious shortcoming of the similarity modification approach is that its effect is brought to bear only through the use of the hierarchical clustering algorithms. It is not driven by a modification to the underlying graph itself (hence would not affect the results of the interchange algorithm, for instance). Another shortcoming is that its impact is not reflected in the decomposition goodness measure M, since M depends only on the underlying graph structure and the particular decomposition at hand, not on the inter-node similarities. The only real impact of this approach is to guide the clustering process along a (possibly) different, presumably better, path than would otherwise be the case. We will see that the second suggested approach, explored next, manages to avoid these drawbacks.

5.1.2 Modification of the Graph Structure.

The major drawbacks to the first approach to incorporating ISR information into the decomposition process hinged on the fact that only the similarity coefficients, not the underlying graph structure, were impacted.

If we study the underlying structure, it is clear that what is needed is a modification that will transform each ISR into a mechanism that serves to more strongly "hook together" the corresponding requirements nodes than would otherwise occur. A simple solution is to transform each ISR into a new graph node, with links to each connected requirement node. Since these new nodes do not represent original requirements, but rather ISR's, they are termed "ISR-nodes." An example of an ISR-node is given in Figure 5.3.

The ISR-node approach does meet all the important requirements for incorporating ISR information into the analysis:

a) since ISR-coupled nodes are now more strongly bound together, through the medium of the new ISR-nodes, decompositions will be more likely to group such nodes together than otherwise;

b) since this technique modifies the underlying graph structure, all decomposition methods, including the interchange technique, continue to apply;

c) the decomposition objective function will reflect the impact of ISR information.

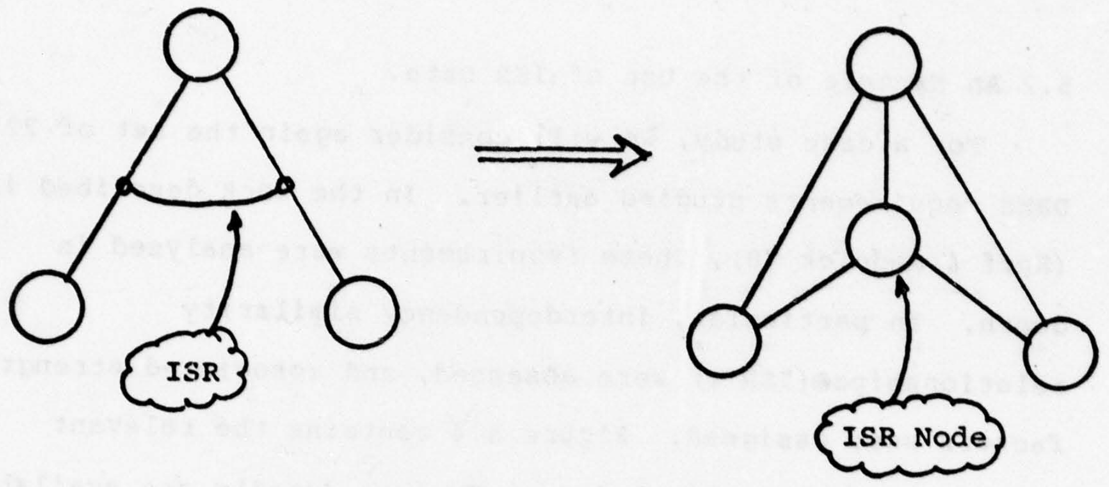


Figure 5.3 (a)

Modification of a 3-node subgraph to include an ISR node.

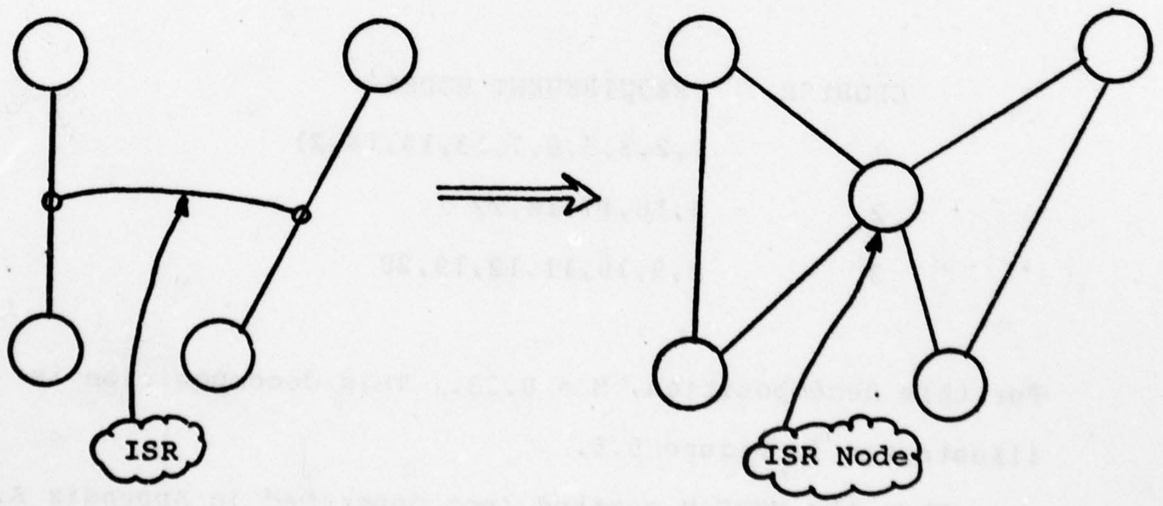


Figure 5.3 (b)

Modification of a 4-node subgraph to include an ISR node.

The ISR-node approach seems the preferable method, and this is borne out in a case study, reported next.

5.2 An Example of the Use of ISR Data.

For a case study, we will consider again the set of 22 DBMS requirements studied earlier. In the work described in (Huff & Madnick 78), these requirements were analyzed in depth. In particular, interdependency similarity relationships (ISR's) were assessed, and associated strength factors were assigned. Figure 5.4 contains the relevant information in graphical form. Further details are available in the reference given above.

In the first part of the test, the original graph structure (with no ISR-nodes) was input to the SDM analysis package, and the best decomposition located using only the clustering algorithms. This turned out to be:

CLUSTER	REQUIREMENT NODES
1	1,2,3,5,6,7,13,14,15,21
2	4,16,17,18,22
3	8,9,10,11,12,19,20

For this decomposition, M = 0.33. This decomposition is illustrated in Figure 5.5.

Then the MODSIM command (not described in Appendix A.1, as it is still experimental) was executed, and the appropriate similarity coefficients modified as discussed earlier. A second decomposition analysis resulted in a

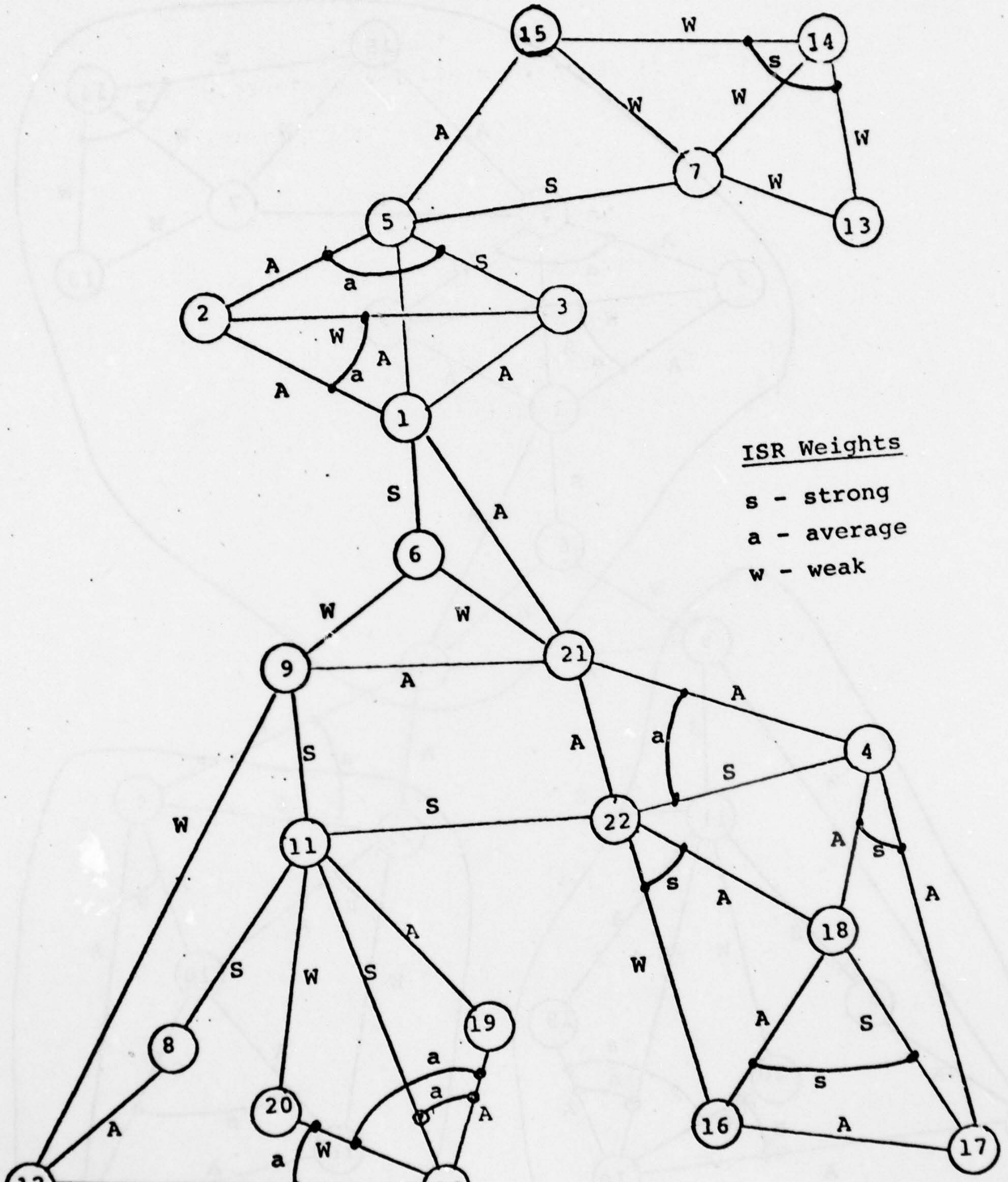


Figure 5.4

The 22-node requirements graph including ISR's

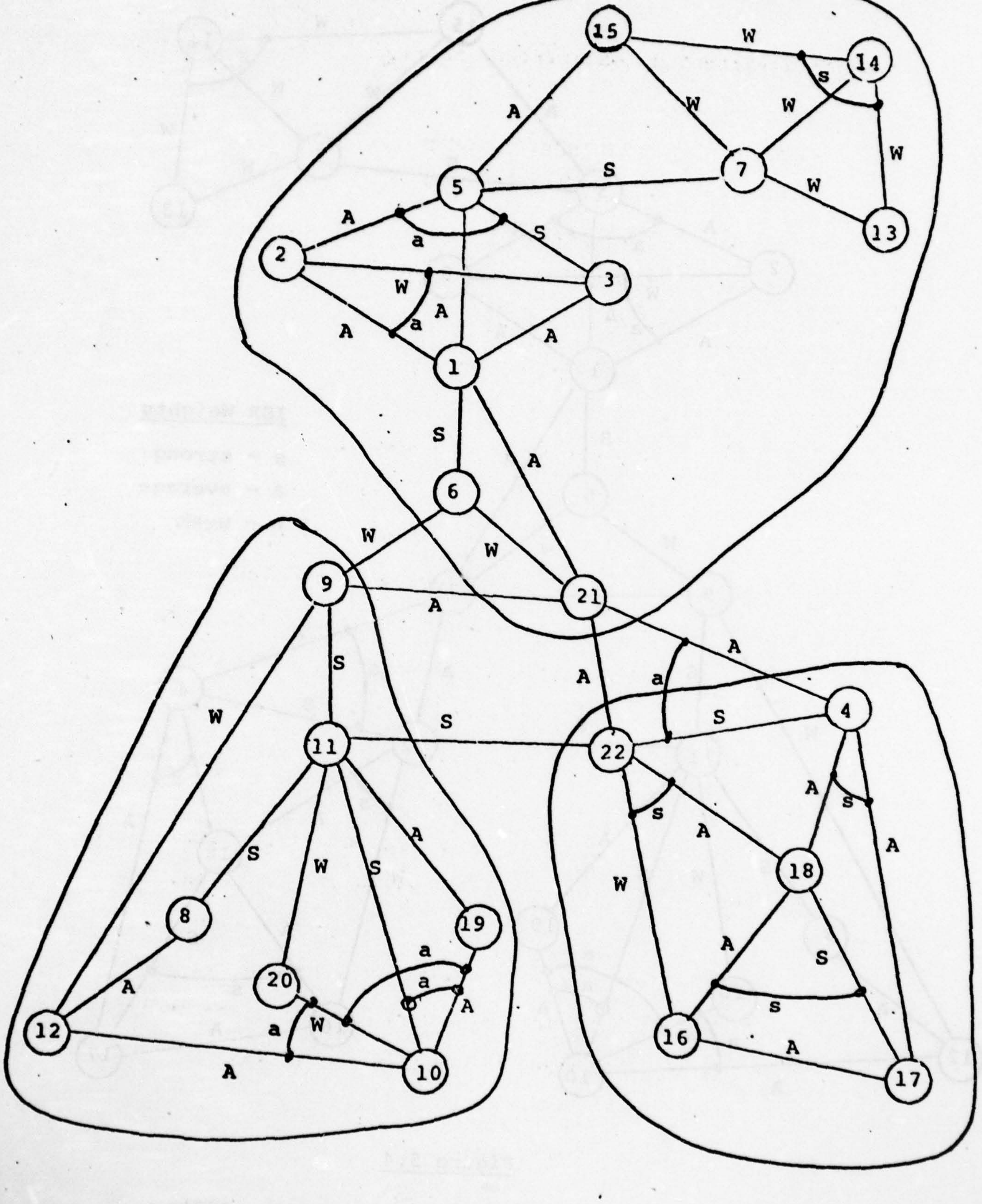


Figure 5.5

Best cluster decomposition of 22-node graph with ISR's not included.

somewhat different decomposition, namely:

CLUSTER	REQUIREMENT NODES
1	1,2,3,5
2	7,13,14,15
3	6,9,21
4	4,16,17,18,22
5	8,10,11,12,19,20

The goodness measure turned out to be M = 0.20. The decomposition is shown graphically in Figure 5.6.

These results are reasonable and believable, as is also suggested by Figure 5.4. The inclusion of the assessed ISR's ought to move the optimal decomposition in the direction of four or five "clumps," rather than the three obtained earlier. However, since the underlying graph was not modified, it is not surprising that M is somewhat lower in the second case.

The second test involved changing the graph structure to include 10 new ISR nodes, then analyzing the resulting 32-node graph in the usual manner. The optimal result obtained from this analysis is slightly different (and, we will argue, somewhat preferable) from that obtained in the first (similarity modification) analysis. It is:

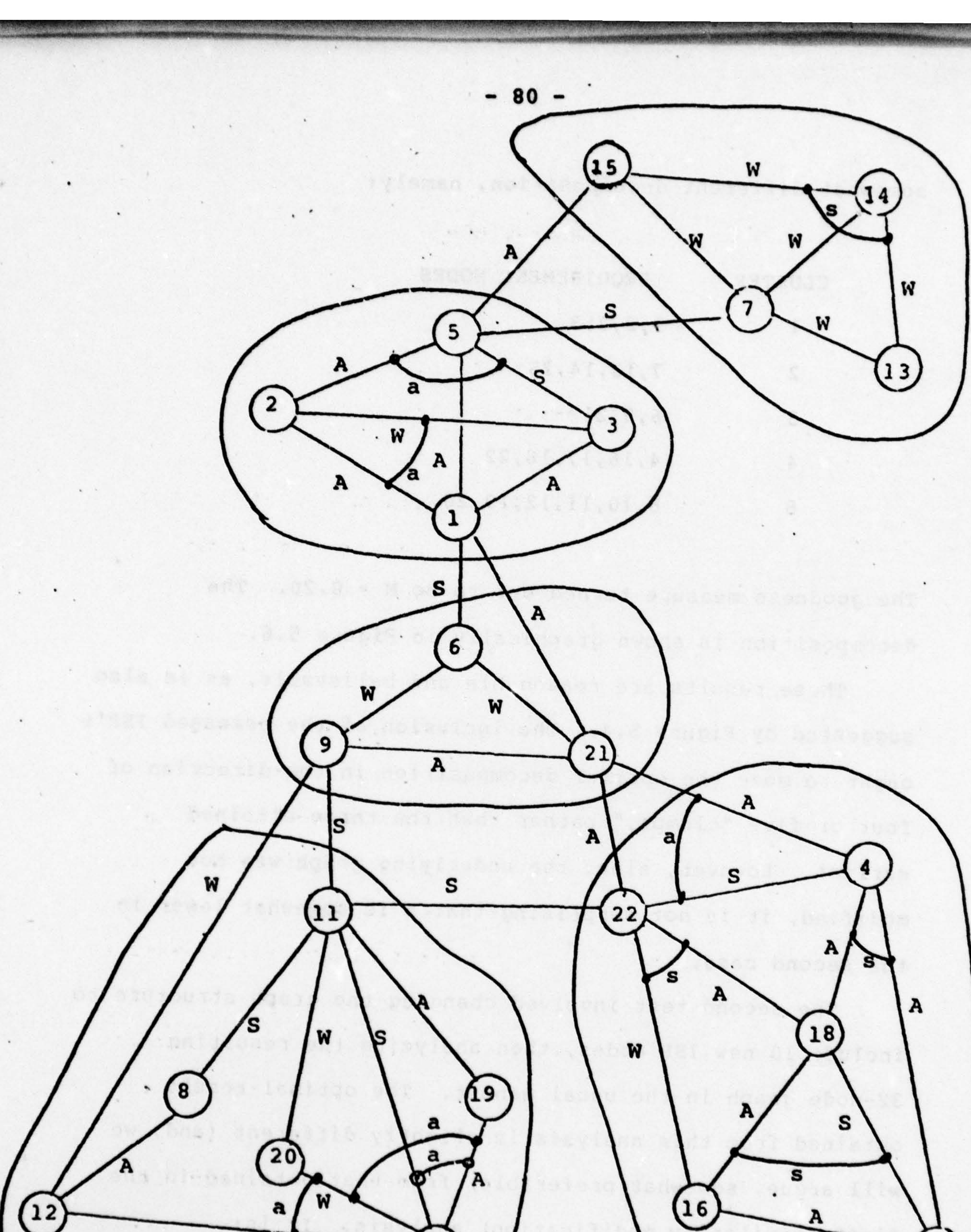


Figure 5.6

Best cluster decomposition of 22-node graph using the similarity modification approach to treating ISR's.

CLUSTER	REQUIREMENT NODES
1	1,2,3,5,6
2	7,13,14,15
3	4,6,16,17,18,21,22
4	8,9,10,11,12,19,20

with M = 0.58. This decomposition is shown in Figure 5.7.

First of all, this new M is higher because it is calculated with respect to the 32-node graph, not the 22-node version. Thus, it is not really comparable to the earlier value.

Secondly, the optimal decomposition in this case includes four, not five, clusters. From the graphs (Figures 5.6 and 5.7), one would have a rather hard time determining which decomposition - the four-subgraph one or the five-subgraph one - was better by inspection. However, it is seen that the above four-cluster decomposition manages to keep together nodes 21, 22, and 4, which were jointly linked by an ISR node as well as linked pairwise by normal interdependencies. If it weren't for the additional ISR links, there would be little to judge between the two alternatives. The fact that the presence of the ISR node swings the balance to four clusters rather than five supports our a priori assessment in this case.

Thus it may be concluded that the ISR-node technique seems to lead toward the desired effect at least as well as the similarity modification technique, and at the same time

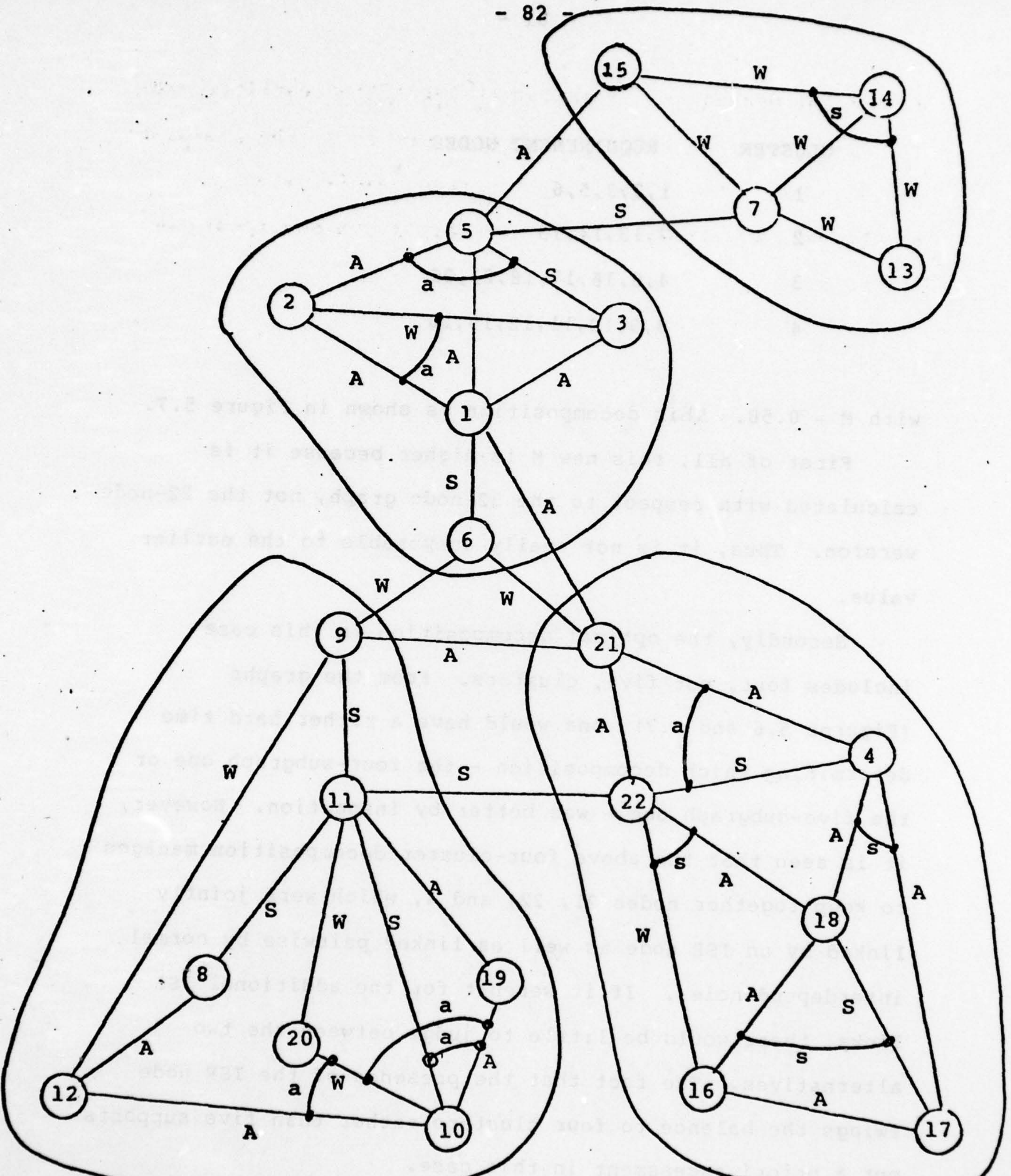


Figure 5.7

Best decomposition of the 22-node graph using ISR-node approach to treating ISR data.

avoids the drawbacks of the latter identified earlier. The ISR-node technique may then be concluded to be the preferable method of incorporating interdependency similarity relationship assessments into the formal SDM decomposition analysis.

6 Conclusions.

Central to the Systematic Design Methodology is the graph model used to represent the design-relevant information pertaining to a target system. A key issue in the development of this methodology is the determination of what information ought to be elicited from a system architect to use in the creation of a preliminary design. The issue is essentially one of cost effectiveness: what is the cost (primarily in designer time and effort) of attempting to elicit a particular piece of information, and what can such information add to the quality of the design?

While easily posed, this question is, at this point, impossible to answer precisely. While neither costs nor benefits are easily assessed, the really difficult question resides in determining benefit: design quality, and in particular, the impact that certain information may have on design quality.

The approach followed within the SDM project has been to use the surrogate "high strength-low coupling decomposition of the system requirements graph" for the real objective, "high design quality." While there are some very believable arguments supporting the appropriateness of this surrogate ((Alexander 64), (Andreu 78)), the case is far from complete. Of course, the same can be said for countless other developments wherein the cost of full-scale objective testing is prohibitively high (including essentially all other software design and development methodologies).

In this research, then, we have placed considerable faith in our own intuition and judgment for effecting a reasonable tradeoff between what design-relevant information may be elicited from designers at a reasonable "cost" and what information is most useful and effective in creating a better preliminary problem structuring. To that end, certain extensions to the basic binary-link representational model used in earlier SDM studies, have been proposed and examined. In this report, the SDM analysis mechanisms were also extended, to incorporate two of the most important model extensions: interdependency strength assessments, and interdependency similarity relationships. The manner in which these extensions impact the SDM decomposition goodness measure M, the inter-requirements similarity calculations, the clustering algorithms, and other aspects of the analysis scheme, has been described, with examples, herein. As well, the appendices include initial documentation of the SDM analysis package.

REFERENCES

Alexander, C.: Notes on the Synthesis of Form, Harvard University Press, 1964.

Anderberg, M.: Cluster Analysis for Applications, Academic Press, 1973.

Andreu, R.: "A Systematic Approach to the Design and Structuring of Complex Software Systems," PhD Dissertation, Sloan School of Management, MIT, February 1978.

Chen, P.: "The Entity-Relationship Model - Towards a Unified View of Data," ACM Trans. on Database Systems, vol. 1, no. 1, March 1976.

Estabrooke, G.: "A Mathematical Model in Graph Theory for Biological Classification," J. Theoretical Biology, vol. 12, no. 2, 1966.

Gottleib, C., and S. Kumar: "Semantic Clustering of Index Terms," Journal of the ACM, vol. 15, no. 4, October 1968.

Hartigan, J.: Clustering Algorithms, John Wiley, 1975.

Hoffer, J. and D. Sevarance: "The Use of Cluster Analysis in Physical Data Base Design," Technical Report 267, Department of Operations Research, Cornell University, August 1975.

Hubert, L.: "Some Applications of Graph Theory to Clustering," Psychometrika, vol. 39, no. 3, September 1974.

Huff, S., and S. Madnick: "An Extended Model for a Systematic Approach to the Design of Complex Systems," Technical Report no. 7, Sloan School of Management, MIT (NTIS no. AO58565), July 1978.

Huff, S.: Decomposition of Weighted Graphs Using the Interchange Partitioning Technique," Technical Report no. 8, Sloan School of Management MIT, Jan. 1979.

McCormack, W. et. al.: "Problem Decomposition and Data Base Reorganization Using a Clustering Technique," Operations Research, vol. 20, no. 5, September 1972.

Myers, G.: Composite/Structured Design, Van Nostrand Reinhold, 1978.

Naur, P., and B. Randell: Software Engineering, Nato Scientific Affairs Division, Brussels, 1968.

Peters, L., and L. Tripp: "Comparing Software Design Methodologies," Datamation, vol. 23, no. 11, November 1977.

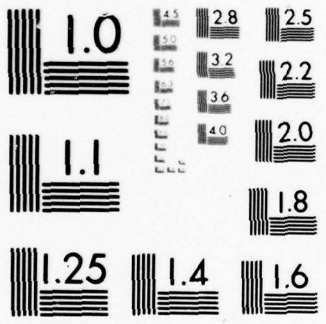
Sangiovanni-Vincentelli, A., et. al.: "An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks," I.E.E.E. Trans. on Circuits and Systems, vol. 24, no. 12, Dec 1977.

Stevens, J., et. al.: "Structured Design," I.B.M. Systems Journal, vol. 13, no. 2, April 1974.

AD-A071 647   ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
ANALYSIS TECHNIQUES FOR USE WITH THE EXTENDED SDM MODEL.(U)
MAY 79 S L HUFF, S E MADNICK   N00039-78-G-0160
UNCLASSIFIED   CISR-P010-7905-09   NL

2 OF 2
AD
A071647

END
DATE
FILMED
8-79
DDC


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

APPENDIX A

SDM Analysis Package Documentation

The analysis program developed for decomposing and analyzing SDM graphs is written in IBM PL/1, and operates within the VM/370-CMS environment. The package is presently run on the MIT 370/168 computer system. While the package is still being tested, and may be slightly modified in the future, it is stable enough in its general characteristics to warrant some brief documentation, as provided here.

General features of the analysis package include:

a) it has been implemented according to the general precepts of structured programming; all subprograms are constrained in size, and have relatively simple structure;

b) the driving routines are command-driven, hence easily extensible (e.g., additional new types of decomposition algorithms could be easily added to the system);

c) subprograms are functionally specific; the relationship between caller and calling routines is always clear and easy to understand;

d) effective use of PL/1's range of available data types has been made in representing clearly and succinctly the key databases used within the system.

The package consists of a master program (named, naturally, MASTER), together with a set of subprograms to implement the various functions. The MASTER program is command-driven, so the user executes various commands by typing the command name (followed in some cases by additional information, after secondary prompting by the system). The various commands, which will be explained in more detail shortly, are:

COMMAND	FUNCTION
STOP	Terminate session, return to the CMS environment;
READGRAP	Read structure information for a graph to be analyzed, from a previously established disk file;
PRINTADJ	Print the adjacency matrix for the graph to the terminal or to the line printer;
CALCSIM	Calculate a similarity matrix for the current graph;
PRINTSIM	Print the similarity matrix to the terminal or the line printer;
SAVESIM	Write the similarity matrix to a disk file, for later retrieval;
READSIM	Read a previously saved similarity matrix from a disk file;
CLUSTER	Execute one of the four hierarchical clustering routines on the current similarity matrix;
READCLUS	Read the trace of the clustering from the file written during the execution of the CLUSTER command;
PRINTMEA	Calculate and type at the terminal or the line printer the goodness measure for specified stages in the clustering trace;
PRINTCLU	Type at the terminal or line printer the node clusters for certain specified stages in the clustering trace;
MODIFY	Make incremental changes to the clustering at a certain stage in the clustering trace;
INTERCH	Execute the interchange partitioning algorithm (a series of subcommands are issued).

A control feature built into the analysis package checks the logical consistency of each command. For example, attempting to execute a SAVESIM or PRINTSIM prior to having calculated a similarity matrix (i.e., to having issued a CALCSIM comamnd) causes a status error. A status error message is issued, and execution continues.

A.1 More on the MASTER Commands.

We consider now each of the above commands, in somewhat more detail.

(1) STOP. This command requires no additional explanation. However, it should be noted that the master program has an attention interrupt trap: hitting the "attention" (or "break") key causes the program to stop its current activity and request a new command. Thus STOP is really the only way of "gracefully" exiting from the MASTER routine (the other way being to hit the attention key multiple times, causing a forced transfer to the CP environment, not a recommended practice).

(2) READGRAP. A typical analysis session begins with reading in a particular graph structure via the READGRAP command. The graph data must have been previously set up in a standard CMS disk file. The first entry must be the number of nodes in the target graph. The remaining entries are to be of the form

(n1,n2,weight),

to indicate that the nodes n1 and n2 are linked in the requirements graph by an interdependency with an associated strength value of "weight." Thus an illustrative graph structure file for a 6-node graph might appear as:

```
6
1 2 .5
1 3 .8
1 4 .8
3 4 .5
2 5 .3
2 6 .4
5 6 .8
```

The order of n1 and n2 in a particular entry is immaterial. The entries need not be on separate logical lines in the file, although in practice it is easier to enter them that way. The graph structure file, as for all files used by the package, must be given a CMS name; for instance, the name "GRAPH DATA A" might be used to identify this file to CMS, whereas the name "GRAPH" could be used for referencing the same file within the PL/1 routines.

Other files that play a role in the execution of MASTER include: (1) the file on which the similarity matrix may be saved for later use (this file is currently named "TEMPSIM"); (2) the intermediate file on which the clustering trace is written during execution of the CLUSTER command (named CLTRACE); (3) a file named "OPTCLUS" on which an optimum cluster vector may be written for later use by other standalone routines.

(3) PRINTADJ and PRINTSIM. These commands use a common subprogram to print out either the adjacency or similarity matrices, either to the user's terminal or to the high-speed line printer located in the main computer facility. The particular device to be used (terminal, or line printer) is determined by a code digit that the user types in response to a follow-up system prompt message.

The matrices are labelled, and the rows and columns numbered appropriately. If the matrix is larger than 15x15, it is "folded" column-wise - i.e., the first 15 columns are printed, followed by the next 15, etc. In all cases, since both matrices are symmetrical, only the lower triangular form is printed.

(4) CALCSIM. This command causes the system to execute a subroutine that calculates all the elements of the similarity matrix, given the weighted adjacency matrix. The calculation is based on the algorithm described in Section 2.2.

(5) SAVESIM, READSIM. These commands write and read, respectively, a temporary copy of the similarity matrix to or from a disk file set up for this purpose. This file is named TEMPSIM within the MASTER routine. These commands make it possible to save the results of a similarity calculation from one session, to be used in a later session that involves the same graph, thereby avoiding the cost of re-calculating the entire matrix.

(6) CLUSTER. This command executes one of the four hierarchical clustering routines described earlier. A secondary prompt requests the user to enter the appropriate "version" number - i.e., "1" for "HIER1", etc. Recall that:

HIER1 - Single linkage,
HIER2 - Complete linkage,
HIER3 - Maximum pre-merge centroid,
HIER4 - Maximum post-merge centroid.

The clustering routine writes a "trace" - that is, a record of the nodal clustering at each step - to an intermediate file named "CLTRACE." As with SAVESIM, this is done in order to provide a "restart" capability for extended or interrupted analysis sessions.

(7) READCLUS. This command reads the clustering trace from the intermediate file "CLTRACE." This can be used to initialize a previously generated clustering trace for further analysis. Of course, it cannot be used until at least one CLUSTER command has been executed, either in the current or an earlier session. It is generally necessary to execute a READCLUS command following each execution of the CLUSTER command; failing to do so may result in performing analysis upon the clustering trace from an earlier CLUSTER calculation.

(8) PRINTMEA. This command can be used to calculate the goodness measure M for one or a consecutive series of steps

in the currently active clustering trace. The actual pass or passes, as well as the output device to be used (terminal or line printer) are entered following secondary promptings.

For example, suppose a 40-node graph had been CLUSTERed, giving rise to a clustering trace consisting of 40 steps, or "passes." The first pass corresponds to each node as a separate cluster; the 40th pass corresponds to a single cluster containing all 40 nodes. In response to a secondary prompt, the user enters the "frompass" and "topass" values (the system checks logical consistency). If only a single pass's measure is required, the user enters that pass number twice in succession. The system responds by calculating and printing out the goodness measure M for each specified pass in the clustering trace, in the form

```
PASS = nn   MEASURE = mm.mmm .
```

(9) PRINTCLU. This command operates in a manner similar to PRINTMEA. However, the clusters themselves, rather than the M values, are printed. A typical printout might appear as:

```
* * * PASS = 8 * * *

CLUSTER 1:  5  6  7
CLUSTER 2:  1  2  3  4
CLUSTER 3:  8  9  10
```

As with PRINTMEA, the user is prompted for device type and for range of passes in the trace for which the clustering information is to be printed.

(10) MODIFY. This command allows the user to make

incremental changes to a current clustering in order to search for local improvements or test out ideas for better decompositions. Following a secondary prompt, the user specifies which pass in the clustering trace he wishes to modify, and the nature of the modifications (which nodes to be placed in which clusters). The resulting decomposition may then be measured, printed, or saved on a special ("OPTCLUS") disk file for later use.

(11) INTERCH. This last command transfers control to the interchange partitioning routine, described next.

A.2 INCHCTL (Interchange Control) Commands.

Most of the primary commands in the analysis package are concerned with setting up the graph data, printing out various data, and executing the clustering routines. This command, however, serves only to pass control to a major subprogram (INCHCTL). This subprogram plays a role somewhat similar to MASTER itself, but with respect to the interchange algorithm.

The interchange technique is described in detail in (Huff 79), and its operational aspects will not be discussed here.

The routine INCHCTL accepts a set of subcommands from the terminal, which allow the user to control step by step the execution if the interchange algorithm. This control is especially useful in performing certain sensitivity analyses during the course of the algorithm's execution.

The commands available within INCHCTL include:

COMMAND	FUNCTION
RETURN	Return control to MASTER;
INITIAL	Initailize the current and proposed partitions;
TYPECUR	Print the current partition at the user's terminal;
TYPEPRO	Print the proposed partition at the user's terminal;
EVALCUR EVALPRO	Evaluate the current/proposed partitions and print the M value at the terminal;
UPDATECU	Replace the current partition with the proposed partition;
RESETPRO	Reset the proposed partition to the value of the current partition;
CALCSTR	Calculate the strength of a specific, or of all, subgraphs in the current partition;
SPLIT	Call the interchange algorithm to partition a particular subgraph in the current decomposition, returning the result in the proposed partition.
AUTO	Execute a master control algorithm to automatically decompose the entire graph.

A central concept in the INCHCTL routine is the use of a "current" and a "proposed" partition. The reason for having two potentially different active partitions is to allow tentative actions tc be taken in the partitioning process without making them irrevocable. For instance, a user might wish to try partitioning a particular subgraph under various minimum-size sub-partition constraints, then select the particular version giving the best M value. Since it would be an extreme computational burden to try all possible combinations, INCHCTL is set up to take advantage of the user's ability to "feel" for a good alternative with a few trial-and-error attempts.

While most of the above commands are self-explanatory, a couple diserve further elaboration.

(1) INITIAL. The structure of the target graph is passed to INCHCTL via the parameter list in the subroutine call. However, no information regarding the initialization of the current and proposed partitions is available at the start. Thus the user's first task is to establish an appropriate decomposition initialization. Normally, one would want to begin at the beginning - i.e., with the entire graph treated as a single "subgraph." Alternatively, a user may have some particular partitioning in mind that he would like to "try out," or from which he would like to start the analysis. Thus the INITIAL command produces a secondary prompting message asking the user to choose between initializing the current and proposed partitions "from the beginning," and entering some other initialization of his choice. Format

requirements under the latter option is provided in the prompting message.

(2) CALCSTR. This command may be used to decide which subgraph within the current decomposition to attack next. Since it is often the case that a user will wish to calculate the strength of all the current subgraphs, a secondary prompt asks him to select a specific subgraph or to specify "all the subgraphs."

Normally, as discussed in (Huff 79), it would make most sense to choose the subgraph with the lowest strength to partition next, although the user may wish to try other alternatives (e.g., the largest subgraph) in some cases.

(3) SPLIT. This command produces two secondary prompts. The first asks the user to enter the identification index of the subgraph he wishes to partition (as printed out by TYPECUR); the second asks for the minimum desired subgraph size for the two subgraphs that will be produced by the interchange algorithm.

(4) AUTO. This command invokes the automatic master control procedure for stepping the interchange algorithm through an entire decomposition. A single secondary prompt asks the user to enter the minimum subgraph size (nmin) to be accepted. The program then proceeds to decompose the target graph by always selecting for the next split

that subgraph with minimum strength and cardinality no less than twice the selected minimum subgraph size.

Appendix B includes an illustration of the use of the interchange control procedure and the automatic governor.

A.3 Routines Included in the Analysis Package.

There are presently a total of 26 individual programs that make up the analysis package. Furthermore, a total of six different files (counting terminal and line printer as "files") may be read or written (or both) during a session. The logical relatiuonships among the programs and files is illustrated in Figure A.1.

The purpose of each program and file is briefly stated below.

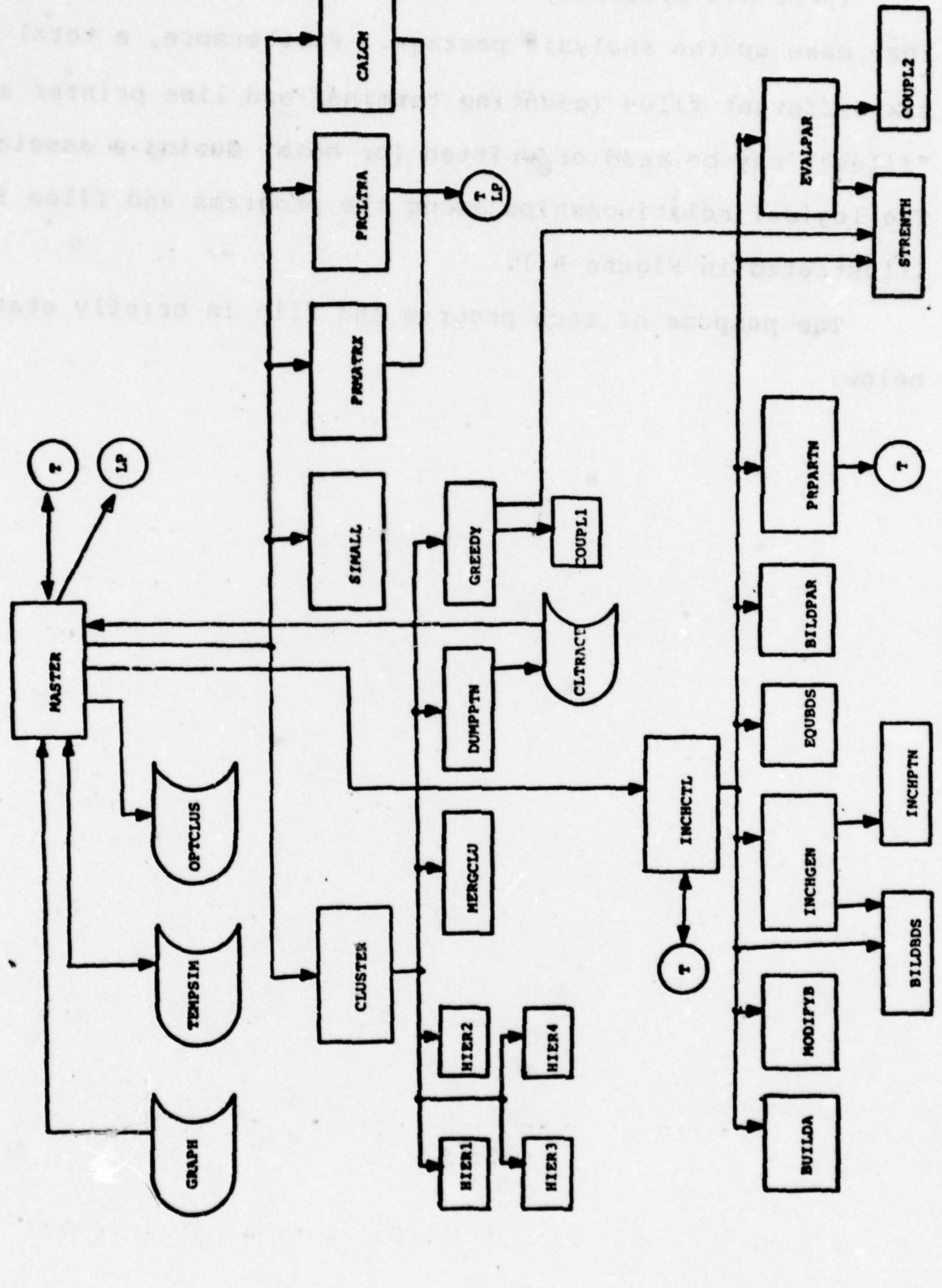


Figure A.1

Control relationships for the routines of the SDM Analysis Package

A.3.1 Programs.

(1) MASTER. Command-driven master control routine (discussed earlier).

(2) SIMALL. Calculates the similarity matrix.

(3) PRMATRX. Prints the similarity or adjacency matrices to line printer or terminal.

(4) CLUSTER. Controls the execution of the clustering algorithm.

(5) DUMPPTN. Called by CLUSTER to write the clustering trace to the intermediate file "CLTRACE."

(6) CALCM. Calculates and prints the decomposition goodness measure for a given step in the clustering trace.

(7) EVALPAR. Calculates the decomposition goodness measure.

(8) STRENTH. Calculates the internal strength of a subgraph.

(9) COUPL2. Calculates the coupling index between two specified subgraphs.

(10) PRCLTRA. Prints the node clustering for a given step in the clustering trace.

(11) HIER1. Performs hierarchical clustering using single linkage.

(12) HIER2. Performs hierarchical clustering using complete linkage.

(13) HIER3. Performs hierarchical clustering using largest pre-merge centroid.

(14) HIER4. Performs hierarchical clustering using largest post-merge centroid.

(15) GREEDY. Performs hierarchical clustering following the

"greedy" algorithm (see Section 3.3).

(16) COUPL1. Calculates the coupling index between a particular subgraph and all other subgraphs that connect to it (for use in the "GREEDY" calculation).

(17) MERGCLU. Merges two specified subgraphs together.

(18) INCHCTL. Controls execution of the interchange algorithm (see Appendix A.2).

(19) BILDBDS. Converts from the vector form for storing partition information to the structure form.

(20) EQUBDS. Equates one partition database to another (used by the UPDATECU and RESETPRO commands within INCHCTL).

(21) BILDPAR. Converts from structure form for storing partition information to the vector form.

(22) BUILDA. Creates a temporary adjacency matrix from a specified subgraph.

(23) MODIFYB. Updates the structure form to reflect a particular cluster merge decision.

(24) PRPARTN. Prints clustering data to terminal or line printer.

(25) INCHGEN. Generates different starting partitions for use in the interchange algorithm routine.

(26) INCHPTN. Performs the interchange calculations to partition a given subgraph.

A.3.2 Files.

(1) User's terminal.

(2) Line printer. Data sent to the line printer is sent directly, as opposed to having it written to a disk file for later spooling. This feature can be easily modified by changing a line in the driving EXEC.

(3) GRAPH. This is the name used within the programs for the file containing the graph structure data. Format for this data was explained in Appendix A.1.

(4) TEMPSIM. This file is used to store the temporary similarity matrix if desired.

(5) CLTRACE. The clustering trace is written to this file, and must be read in before executing commands dealing with the results of a particular clustering.

(6) OPTCLUS. During the execution of the MODIFY command, the user is asked whether he wishes to save the modified decomposition he has created. If he elects to save it, it will be written to this file.

APPENDIX B

Terminal Execution Trace.

This appendix consists of the terminal execution trace of a sample session using the SDM analysis package. The graph being analyzed is the 22-node DBMS requirements graph referred to in Section 4 (see Figure 4.1).

The commentary in italics was added to amplify and explain the trace data.

Explanatory comments

note: underlined parts are user responses.

```
MASTER                                          Execute the MASTER program

EXECUTION BEGINS...

OK:.READGRAP                                    Read graph data from disk

GRAPH DATA ENTERED. NNODE= 22

OK:.PRINTADJ                                    Print the adjacency matrix to the
                                                line printer
PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.0

OK:.WRONGCOM                                    Unrecognised command

BAD COMMAND.

OK:.CALCSIM                                     Calculate the similarity matrix
                                                and print it at the terminal

OK:.PRINTSIM

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1
```

```
SIMILARITY MATRIX:

          1       2       3       4       5       6       7       8       9      10

   1  1.000
   2  0.281   1.000
   3  0.315   0.450   1.000
   4  0.044   0.000   0.000   1.000
   5  0.256   0.327   0.360   0.000   1.000
   6  0.295   0.151   0.134   0.031   0.092   1.000
   7  0.093   0.141   0.188   0.000   0.256   0.000   1.000
   8  0.000   0.000   0.000   0.000 !

ATTENTION INTERRUPT.
```

Printout halted using 'break' key

```
OK:.CLUSTER

SELECT PROCEDURE - 1/2/3...
:
.1
```

Perform hierarchical clustering using HIER1

```
OK:.PRINTMEA

STATUS ERROR.  CLUSTERING TRACE NOT YET ESTABLISHED.

OK:.READCLUS
```

Clustering trace must be read from disk prior to analyzing .

```
OK:.PRINTMEA

SELECT RANGE, LOWER TO HIGHER.
:
.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1
.

PASS=   18   MEASURE=   -0.1487

PASS=   19   MEASURE=   -0.0996

PASS=   20   MEASURE=   -0.1870

PASS=   21   MEASURE=    0.0280

PASS=   22   MEASURE=    0.0373
```

Measures for various stages in the clustering. Best M = 0.0373, corresponding to a single cluster)

```
OK:.CLUSTER

SELECT PROCEDURE - 1/2/3...
:
.2
```

Repeat using HIER2

```
OK:.READCLUS

OK:.PRINTMEA

SELECT RANGE, LOWER TO HIGHER.
:
.16 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1

PASS=   16   MEASURE=   -0.7122

PASS=   17   MEASURE=   -0.6049

PASS=   18   MEASURE=    0.0602

PASS=   19   MEASURE=    0.2802

PASS=   20   MEASURE=    0.1234

PASS=   21   MEASURE=    0.0280

PASS=   22   MEASURE=    0.0373
OK:.PRINTCLU

SELECT RANGE, LOWER TO HIGHER.
:
.19 19

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1


*** PASS  19 ***

CLUSTER    1:    1    2    3    5    6    7   15
CLUSTER    2:    4   16   17   18   21   22
CLUSTER    3:    8    9   10   11   12   19   20
CLUSTER    4:   13   14

OK:.CLUSTER
```

Best measure at pass 19 (four clusters)

Examine the clustering at pass 19

Repeat clustering using HIER3

```
SELECT PROCEDURE - 1/2/3...
:
.3

OK:.READCLUS

OK:.PRINTMEA

SELECT RANGE, LOWER TO HIGHER.
:
.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1

PASS=    18    MEASURE=     0.2958

PASS=    19    MEASURE=     0.3034

PASS=    20    MEASURE=     0.3253

PASS=    21    MEASURE=     0.1170

PASS=    22    MEASURE=     0.0373
```

Best results at pass 20 - better M than for previous techniques.

```
OK:.PRINTCLU

SELECT RANGE, LOWER TO HIGHER.
:
.18 21

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1
```

Print the clustering for a range of steps in the clustering trace

```
*** PASS  18 ***

CLUSTER    1:    1    6   21
CLUSTER    2:    2    3    5    7   15
CLUSTER    3:    4   16   17   18   22
CLUSTER    4:    8    9   10   11   12   19   20
CLUSTER    5:   13   14

*** PASS  19 ***

CLUSTER    1:    1    2    3    5    6    7   15   21
CLUSTER    2:    4   16   17   18   22
CLUSTER    3:    8    9   10   11   12   19   20
CLUSTER    4:   13   14

*** PASS  20 ***

CLUSTER    1:    1    2    3    5    6    7   13   14   15   21   This clustering
CLUSTER    2:    4   16   17   18   22                               corresponds to the
CLUSTER    3:    8    9   10   11   12   19   20                     best M value

*** PASS  21 ***

CLUSTER    1:    1    2    3    5    6    7   13   14   15   21
CLUSTER    2:    4    8    9   10   11   12   16   17   18   19   20   22

OK:.CLUSTER

SELECT PROCEDURE - 1/2/3...
:                                                 Repeat using HIER4
.4

OK:.READCLUS

OK:.PRINTMEA

SELECT RANGE, LOWER TO HIGHER.
:
.18 22

PRINT TO TERMINAL(1) OR LINE PTR(0)?
:
.1
```

```
PASS=   18   MEASURE=     0.0374

PASS=   19   MEASURE=     0.1643

PASS=   20   MEASURE=     0.2330
```

Best M at pass 20 - not as good as for HIER3

```
PASS=   21   MEASURE=     0.1256

PASS=   22   MEASURE=     0.0373

OK:.MODIFY
```

Try new clustering arrangement

```
WHICH PASS TO BE MODIFIED?
:
.20

ENTER CHANGES, IN FORMAT: NODE NO.,CLUS NO.   TERMINATE WITH 0,0
:
.7 4 13 4 14 4 15 4 0 0


*** PASS  20 ***

CLUSTER    1:   1   2   3   5   6
CLUSTER    2:   4  16  17  18  21  22
CLUSTER    3:   8   9  10  11  12  19  20
CLUSTER    4:   7  13  14  15
```

Results are better than anything so far (M = .378)

```
PASS=   20   MEASURE=     0.3782
WANT TO SAVE? (YES=1, NO=0)
:
.0
```

Could save this clustering on disk if so desired

```
OK:.INTERCH
```

Enter the "interchange algorithm" control routine

```
   INCH:.INITIAL
```

Initialize to standard single subgraph

```
   SELECT NORMAL INIT. (1) OR OWN PARTITIONING (0)
:
.1

   ECHO OF INITIAL PARTITION:
   # 1:   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15

   INCH:.SPLIT
```

Partition the starting subgraph

```
   ENTER SUBGRAPH ID NUMBER.
:
.1
```

```
     CARDINALITY OF SUBGRAPH NUMBER  1 =    22
     ENTER MINIMUM SUBGRAPH SIZE.
 :
 .1
 ..
                                                   Resulting partition
     INCH:.TYPEPRO

     # 1:    1    2    3    5    6    7   13   14   15
     # 2:    4    8    9   10   11   12   16   17   18   19   20   21   22

     INCH:.EVALPRO

     M FOR PROPOSED DECOMPOSITION =  0.133              M value for this
                                                        partitioning
     INCH:.UPDATECU

     CURRENT PARTITION <- PROPOSED PARTITION.           Set the current partitic
                                                        to this proposed one
     INCH:.CALCSTR

     ENTER SUBGRAPH ID NUMBER.  ENTER 0 FOR "ALL".      Calculate strength for
 :                                                      both subgraphs in
 .0                                                     current partition


     STRENGTH FOR SUBGRAPH  1 =      0.073

     STRENGTH FOR SUBGRAPH  2 =      0.068              Subgraph 2 lower - selec
                                                        it for next partitioning
     INCH:.SPLIT

     ENTER SUBGRAPH ID NUMBER.
 :
 .2

     CARDINALITY OF SUBGRAPH NUMBER  2 =    13
     ENTER MINIMUM SUBGRAPH SIZE.
 :
 .1


     INCH:.TYPEPRO

     # 1:    1    2    3    5    6    7   13   14   15   Result after partitionin
     # 2:    4   16   17   18   21   22                  subgraph 2
     # 3:    8    9   10   11   12   19   20

     INCH:.EVALPRO


     M FOR PROPOSED DECOMPOSITION =  0.299              M getting better

     INCH:.UPDATECU
                                                        Set current partition to
                                                        this proposed one
```

```
CURRENT PARTITION <- PROPOSED PARTITION.

INCH:.CALCSTR

ENTER SUBGRAPH ID NUMBER.  ENTER 0 FOR "ALL".
:
.1

STRENGTH FOR SUBGRAPH  1 =     0.073

INCH:.CALCSTR

ENTER SUBGRAPH ID NUMBER.  ENTER 0 FOR "ALL".
:
.0

STRENGTH FOR SUBGRAPH  1 =     0.073
STRENGTH FOR SUBGRAPH  2 =     0.177
STRENGTH FOR SUBGRAPH  3 =     0.097

INCH:.SPLIT

ENTER SUBGRAPH ID NUMBER.
:
.1

CARDINALITY OF SUBGRAPH NUMBER  1 =    9
ENTER MINIMUM SUBGRAPH SIZE.
:
.4

INCH:.TYPEPRO

# 1:   1   2   3   5   6
# 2:   4  16  17  18  21  22
# 3:   8   9  10  11  12  19  20
# 4:   7  13  14  15

INCH:.EVALPRO

M FOR PROPOSED DECOMPOSITION =  0.378

INCH:.UPDATECU
```

Calculate strengths for the three subgraphs

Partition subgraph 1 (lowest strength)

Resulting proposed partition

Best M so far of all approaches tried

```
CURRENT PARTITION <- PROPOSED PARTITION.

INCH:.EVALCUR

M FOR CURRENT DECOMPOSITION =  0.378
```

Cal evaluate current partition also

```
INCH:.CALCSTR

ENTER SUBGRAPH ID NUMBER.  ENTER 0 FOR "ALL".
:
.0

STRENGTH FOR SUBGRAPH  1 =       0.163
```

Repeat procedure one more cycle

```
STRENGTH FOR SUBGRAPH  2 =       0.177

STRENGTH FOR SUBGRAPH  3 =       0.097

STRENGTH FOR SUBGRAPH  4 =       0.067

INCH:.SPLIT

ENTER SUBGRAPH ID NUMBER.
:
.3

CARDINALITY OF SUBGRAPH NUMBER  3 =   7
ENTER MINIMUM SUBGRAPH SIZE.
:
.3


INCH:.TYPEPRO

# 1:    1   2   3   5   6
# 2:    4  16  17  18  21  22
# 3:    9  10  11  19
# 4:    7  13  14  15
# 5:    8  12  20
```

Proposed partition

```
INCH:.EVALPRO

M FOR PROPOSED DECOMPOSITION =  0.041
```

M way down from previous partitioning (don't accept)

Try setting up own partition.

```
INCH:.INITIAL

SELECT NORMAL INIT. (1) OR OWN PARTITIONING (0)
:
.0

ENTER LIST OF NODES FOR EACH CLUSTER IN TURN.
TERMINATE EACH LIST WITH A ZERO.
TERMINATE ENTIRE ENTRY WITH ANOTHER ZERO.
:
.1 2 3 5 6 9 21 0

:
.7 13 14 15 0

:
.4 16 17 18 22 0

:
.8 10 11 12 19 20 0 0

ECHO OF INITIAL PARTITION:
# 1:    1   2   3   5   6   9  21
# 2:    7  13  14  15
# 3:    4  16  17  18  22
# 4:    8  10  11  12  19  20

INCH:.EVALCUR

M FOR CURRENT DECOMPOSITION =  0.371
```

Good results, but not as good as earlier.

```
INCH:.AUTO
```

execute automatic governor

```
ENTER MINIMUM SUBGRAPH SIZE.
:
.4

BEGIN AUTO. DECOMPOSITION. USE ATTN TO STOP EARLY.

M FOR FULL GRAPH =   0.037
```

```
NEXT PARTITION IS:
# 1:    4    8    9   10   11   12   16   17   18   19   20   21   22
# 2:    1    2    3    5    6    7   13   14   15

M FOR THIS DECOMPOSITION =  0.133                 result of first partitioning

SUBGRAPH STRENGTHS:

          NO     STREN  CARD MINCARD
           1     0.068   13      8                selects subgraph 1 for
           2     0.073    9      8                next partitioning
          SELECT NO.     1

NEXT PARTITION IS:
# 1:    8    9   10   11   12   19   20
# 2:    1    2    3    5    6    7   13   14   15
# 3:    4   16   17   18   21   22

M FOR THIS DECOMPOSITION =  0.299

SUBGRAPH STRENGTHS:

          NO     STREN  CARD MINCARD
           1     0.097    7      8                note subgraph 3 is ineligible
           2     0.073    9      8                for next split - only has
           3     0.177    6      8                6 nodes (nmin = 4)
          SELECT NO.     2

NEXT PARTITION IS:
# 1:    8    9   10   11   12   19   20
# 2:    1    2    3    5    6
# 3:    4   16   17   18   21   22
# 4:    7   13   14   15
                                                  *** BEST M ***
M FOR THIS DECOMPOSITION =  0.378                 same end result as found
                                                  earlier.
SUBGRAPH STRENGTHS:

          NO     STREN  CARD MINCARD
           1     0.097    7      8
           2     0.163    5      8                no subgraphs elegible now.
           3     0.177    6      8                AUTO stops when all sub-
           4     0.067    4      8                graphs contain fewer than
                                                  2*nmin nodes.
FOUND NO MORE SUBGRAPHS FOR PARTITIONING.
RETURN TO INCHCTL COMMAND LEVEL.

INCH:.RETURN
                                                  return to MASTER

INTERCHANGE ANALYSIS ENDED.                       stop run.

OK:.STOP

                                                  stop run.

RUN ENDED.
R;
```